



**Europäisches  
Patentamt**

**European  
Patent Office**

**Office européen  
des brevets**

**Bescheinigung**

**Certificate**

**Attestation**

Die angehefteten Unterlagen stimmen mit der ursprünglich eingereichten Fassung der auf dem nächsten Blatt bezeichneten europäischen Patentanmeldung überein.

The attached documents are exact copies of the European patent application described on the following page, as originally filed.

Les documents fixés à cette attestation sont conformes à la version initialement déposée de la demande de brevet européen spécifiée à la page suivante.

**Patentanmeldung Nr.    Patent application No.    Demande de brevet n°**

03425081.1

Der Präsident des Europäischen Patentamts;  
Im Auftrag

For the President of the European Patent Office

Le Président de l'Office européen des brevets  
p.o.

**R C van Dijk**



Anmeldung Nr:  
Application no.: 03425081.1  
Demande no:

Anmeldetag:  
Date of filing: 11.02.03  
Date de dépôt:

Anmelder/Applicant(s)/Demandeur(s):

STMicroelectronics S.r.l.  
Via C. Olivetti, 2  
20041 Agrate Brianza (Milano)  
ITALIE

Bezeichnung der Erfindung/Title of the invention/Titre de l'invention:  
(Falls die Bezeichnung der Erfindung nicht angegeben ist, siehe Beschreibung.  
If no title is shown please refer to the description.  
Si aucun titre n'est indiqué se référer à la description.)

A process for translating instructions for an ARM-type processor into  
instructions for a LX-type processor relative translator device and computer  
program product

In Anspruch genommene Priorität(en) / Priority(ies) claimed / Priorité(s)  
revendiquée(s)  
Staat/Tag/Aktenzeichen/State/Date/File no./Pays/Date/Numéro de dépôt:

Internationale Patentklassifikation/International Patent Classification/  
Classification internationale des brevets:

G06F9/00

Am Anmeldetag benannte Vertragsstaaten/Contracting states designated at date of  
filing/Etats contractants désignées lors du dépôt:

AT BE BG CH CY CZ DE DK EE ES FI FR GB GR HU IE IT LU MC NL  
PT SE SI SK TR LI

"Procedimento per tradurre istruzioni per un processore di tipo ARM in istruzioni per un processore di tipo LX, relativo dispositivo di traduzione e prodotto informatico"

5

\* \* \*

Campo dell'invenzione

La presente invenzione si riferisce alle tecniche per la traduzione delle istruzioni destinate a operare su processori diversi. L'invenzione è stata messa a  
10 punto con particolare riferimento alla possibile applicazione alla traduzione di istruzioni eseguibili su un processore di tipo ARM in istruzioni eseguibili su un processore di tipo LX, quale ad esempio il microprocessore ST200-LX di produzione della  
15 Richiedente.

Descrizione della tecnica nota

Un microprocessore ARM è tipicamente un microprocessore scalare pipelined a 32 bits, cioè un microprocessore la cui architettura interna è  
20 costituita da diversi stage logici, ognuno dei quali contiene una istruzione in un ben specifico stato. Detto stato può essere di: caricamento della istruzione stessa dalla memoria, di decodifica, di indirizzamento di un file di registri, di esecuzione, di  
25 scrittura/lettura dati dalla memoria. Il numero di bit si riferisce all'ampiezza dei dati e delle istruzioni sui quali opera. Le istruzioni sono generate in uno specifico ordine mediante compilazione e eseguite nello stesso. Un microprocessore LX è tipicamente un  
30 microprocessore del tipo definito Very Long Instruction Word o WLIW, pipelined a 128 bits. Un microprocessore superscalare pipelined possiede un'architettura interna costituita da diversi stage logici alcuni dei quali possono eseguire istruzioni in parallelo, ad esempio  
35 nella fase di esecuzione. Tipicamente il parallelismo è

di 4 istruzioni a 32 bits ciascuna (eguale a 128 bits) mentre i dati sono espressi in 32 bits.

Il processore si dice superscalare se le istruzioni sono riordinate dinamicamente in fase di esecuzione in modo da alimentare gli stage di esecuzione che potenzialmente possono lavorare in parallelo e se le istruzioni non presentano dipendenze mutue, alterando così l'ordine generato staticamente dalla compilazione del codice sorgente.

Il processore si dice VLIW se invece le istruzioni sono riordinate staticamente in fase di compilazione ed eseguite in quell'ordine fisso, non modificabile in fase di esecuzione.

Per maggiori informazioni circa la architettura dei microprocessori si può fare riferimento alla descrizione riportata nel testo: *Computer Organization & Design The hardware/software interface*, DA. Patterson & J.L. Hennessy, Morgan Kaufmann.

Il processore ARM è una macchina RISC single-issue dotata comunque di un set di modi di indirizzamento sufficientemente ampio (le istruzioni di data-processing supportano ben nove modalità diverse) e presenta la possibilità di eseguire condizionalmente la quasi totalità delle sue istruzioni sulla base dei flag contenuti nel registro di stato denominato CPSR.

Il processore LX è un processore VLIW four-issue, che nel seguito della presente descrizione sarà sempre illustrato nella versione single-cluster. Il processore LX a differenza del processore ARM ha soltanto due modi di indirizzamento (immediato e da registro) e non consente l'esecuzione condizionata, ma data la presenza di quattro lane operanti in parallelo permette di eseguire in parallelo più alternative (con un massimo di 4 istruzioni) per poi effettuare la scelta del risultato opportuno una volta valutata la condizione

sull'esecuzione.

Il microprocessore ARM nella versione 5 cui si farà riferimento nel seguito, possiede un'architettura interna a 32 bit che garantisce uno spazio di indirizzamento di 4 Gbyte e presenta 31 registri di uso generale, dei quali però solo 16, indicati con i riferimenti da R1 a R16, sono accessibili contemporaneamente.

Esistono infatti sette differenti modi di funzionamento necessari per gestire le varie tipologie di eccezioni a cui il processore deve rispondere:

USER	modo di esecuzione normale
FIQ	gestione di interrupt ad alta velocità
IRW	gestione di interrupt di tipo generico

SUPERVISOR modo privilegiato per il sistema operativo

ABORT protezione dell'accesso alla memoria e/o memoria virtuale

UNDEFINED codice di operazione non definito, per l'emulazione di coprocessore

SYSTEM modalità privilegiata per operazioni particolari del sistema operativo.

Due dei 16 registri accessibili hanno un ruolo particolare:

- il registro R15 è utilizzato come program counter (PC) cioè contiene l'indirizzo dell'istruzione da eseguire;
- il registro R14 è utilizzato come link-register (LR); contiene cioè l'indirizzo dell'istruzione da eseguire a seguito del ritorno dall'esecuzione di una subroutine.

Normalmente, inoltre, il registro R13 è utilizzato dal software come stack pointer.

Due o più dei registri di uso generale sono replicati per i vari modi di funzionamento al fine di velocizzare la gestione delle eccezioni.

5 Nei modi IRQ, Abort, Undefined e Supervisor rispetto al modo User sono replicati solo i registri R13 e R14 (ovvero link-register e stack-pointer).

Nel modo FIQ, per rendere ancora più veloce la gestione dell'eccezione sono stati replicati anche i registri da R8 a R12.

10 Il modo System, pur avendo tutti i benefici di un modo privilegiato, vede tutti gli stessi registri del modo User.

Ovviamente il program counter non è replicato in nessuno dei modi.

15 Oltre ai registri di uso generale è disponibile un registro di stato CPSR, il cui contenuto è mostrato in Tabella 1 contenente informazioni sul risultato dell'esecuzione e sul modo di funzionamento.

31	30	29	28	27	26		8	7	6	5	4		0
N	Z	C	V	Q		(RAZ)		I	F	T		MODE	

Tabella 1

20 dove

- N flag (negative flag): N=1 se il risultato di una operazione è negativo;

25 - C flag (carry flag): C=1 se il risultato di una operazione di somma genera riporto oppure se durante la fase di generazione degli operandi per un' operazione logica si sono verificate particolari condizioni, C=0 se il risultato di un' operazione di sottrazione genera riporto (borrow);

30 - V flag (overflow flag): V=1 se una operazione aritmetica ha generato overflow

- Z flag (zero flag): Z=1 se il risultato di un'operazione è zero;

- Q flag : nelle versioni Extended Q=1 se il risultato di una delle operazioni del gruppo Enhanced DSP genera overflow o saturazione.

I bit dal 26 all'8 non devono essere modificati e sono letti come zero.

- I bit: se I =1 disabilita gli interrupt IRQ;
- F bit: se F=1 disabilita gli interrupt FIQ;
- T bit: se T=0 il processore sta eseguendo nella normale modalità ARM, se T=1 è attiva la modalità di esecuzione Thumb. In questa modalità ARM interpreta un set di istruzioni ridotto, con codici operazione o opcode che occupano solo 16 bit ma con aritmetica e registri a 32 bit, e vede contemporaneamente solo 8 registri di uso generale.

I 5 bit meno significativi del registro di stato descrivono il modo di funzionamento del processore ARM, come si può vedere dalla seguente Tabella 2:

CPSR (4:0)	MODO
0b10000	USER
0b10001	FIQ
0b10010	IRQ
0b10011	SUPERVISOR
0b10111	ABORT
0b11011	UNDEFINED
0b11111	SYSTEM

**Tabella 2**

- Tutti i modi privilegiati, oltre al registro CPSR, presentano poi un registro SPSR, replicato per ogni modo. Il registro SPSR associato ad un certo modo viene utilizzato per salvare la parola di stato contenuta nel registro CPSR quando viene sollevata l'eccezione corrispondente a quel modo; alla fine della gestione dell'eccezione il registro CPSR verrà ripristinato con

il valore del registro SPSR. Le istruzioni del processore ARM possono essere classificate in sei gruppi:

- data-processing (modo di indirizzamento 1);
- 5    - load & store di word (32 bit) o unsigned byte (modo di indirizzamento 2);
- load & store di halfword (16 bit) o signed byte (modo di indirizzamento 3);
- load & store multiple (modo di indirizzamento 4)
- 10   - istruzioni per i coprocessori (modo di indirizzamento 5);
- salti.

Il processore ARM consente l'esecuzione condizionata di quasi tutte le sue istruzioni sulla base dei flag N, C,V,Z contenuti nel registro di stato CPSR.

La condizione è descritta nei quattro bit più significativi dell'opcode del processore ARM.

Fanno eccezione l'istruzione BLX (branch, link and exchange to Thumb state) e le istruzioni che fanno riferimento ai coprocessori, che non sono condizionali.

Le varie combinazioni dei flag generano sedici tipi di esecuzione condizionata:

- AL (always): l'istruzione viene sempre eseguita;
- 25   - NV (never: l'istruzione non viene mai eseguita, non è definita oppure fa parte delle istruzioni non condizionali alle quali si è accennato in precedenza;
- EQ(equal): Z=1;
- 30   - NE (not equal): Z=0;
- CS/HS (carry set - unsigned higher or same): C=1;
- CC/LO (carry clear - unsigned lower): C=0;
- MI (minus - negative): N=1;
- 35   - PL (plus - positive or zero): N=0;

- VS (overflow):  $V=1$ ;
- VC (no overflow):  $V=0$ ;
- HI (unsigned higher):  $C=1$  e  $Z=0$ ;
- LS (unsigned lower or same):  $C=0$  o  $Z=1$ ;
- 5 - GE (unsigned greater than or equal):  $N=V$ ;
- LT (signed less than):  $N \neq V$ ;
- GT (signed greater than):  $Z=0$  e  $N=V$ ;
- LE (signed less than or equal):  $Z=1$  o  $N \neq V$ .

Le modalità di indirizzamento del processore ARM  
10 per le istruzioni di data-processing sono undici:

- Immediato;
- diretto da registro;
- shift logico a sinistra da registro (l'entità dello shift è contenuta in un registro);
- 15 - shift logico a sinistra da immediato (l'entità dello shift è espressa da un immediato a 5 bit contenuto nell'opcode);
- shift logico a destra da registro;
- shift logico a destra da immediato;
- 20 - shift aritmetico a destra da registro;
- shift aritmetico a destra da immediato;
- rotazione a destra da registro;
- rotazione a destra da immediato;
- rotazione attraverso il carry flag.

25 Le istruzioni di data-processing sono operazioni di tipo logico o aritmetico che vengono eseguite dall'unità aritmetico logica o ALU a 32 bit del processore ARM.

30 Queste operazioni possono modificare il valore dei flag del registro CPSR sulla base del loro risultato quando il bit 20 (S bit) dell'opcode è a valore alto. La fase di esecuzione di queste operazioni dura sempre un solo ciclo di clock.

35 Il processore ARM è poi in grado di eseguire moltiplicazioni e moltiplicazioni con accumulo di

numeri fino a 32 bit, generando un risultato a 64 bit che viene spezzato in due registri destinazione.

5 Tutte le operazioni di moltiplicazione supportano solo l'indirizzamento diretto da registro e la loro fase di esecuzione dura un solo ciclo di clock, indipendentemente dalla necessità o meno di effettuare l'operazione di accumulo alla fine della moltiplicazione stessa.

10 Le operazioni di load & store in memoria del modo 2 agiscono su word e unsigned byte e supportano nove modi di indirizzamento, che fanno comunque uso di un registro base e di uno spiazzamento:

- registro base +/- immediato a 12 bit;
- registro base +/- registro offset;
- 15 - registro base +/- registro offset scalato (il registro offset è shiftato con modalità analoghe alle istruzioni di data-processing; l'entità dello shift è descritta da un immediato;
- registro base +/- immediato pre-indexed (il
- 20 registro base viene aggiornato prima di effettuare l'accesso a memoria);
- registro base +/- registro offset pre-indexed;
- registro base +/- registro scalato pre-indexed;
- registro base +/- immediato post-indexed (il
- 25 registro base viene aggiornato dopo avere effettuato l'accesso a memoria);
- registro base +/- registro offset post-indexed;
- registro base +/- registro scalato post-indexed.

30 L'operazione di lettura dalla memoria di una parola a 32 bit non necessita che l'indirizzo sia di per sè word-aligned; la lettura viene effettuata comunque, dopo di che la parola o word viene ruotata di 8,16 o 24 se l'indirizzo non era word-aligned ma terminava per 0b01, 0b10 o 0b11.

35 L'operazione di scrittura di una word invece si

autoallinea ignorando completamente i due bit meno significativi dell'indirizzo, quindi non è esattamente il duale dell'operazione di lettura.

Le operazioni di load & store in memoria del modo 3  
 5 agiscono su halfword e signed byte e supportano solo sei dei nove modi di indirizzamento associati al modo 2:

- registro base +/- immediato a 8 bit;
- registro base +/- registro offset;
- 10 - registro base +/- immediato pre-indexed;
- registro base +/- registro offset pre-indexed;
- registro base +/- immediato post-indexed;
- registro base +/- registro offset post-indexed.

Al contrario di quanto accade per le istruzioni del  
 15 modo 2, le operazioni di lettura e scrittura su halfword (16 bit) necessitano di indirizzi halfword-aligned per essere eseguite correttamente.

Le operazioni di load & store multiple del modo 4  
 20 contengono all'interno del loro opcode un campo di 16 bit che marca con un bit a livello alto i registri interessati al trasferimento.

Queste operazioni presentano quattro modi di indirizzamento:

- increment after: la lista di registri è caricata  
 25 in memoria (per le store) o dalla memoria (per le operazioni di load) a partire dall'indirizzo puntato da un registro base. I registri successivi verranno caricati in indirizzi ottenuti incrementando di quattro (dato che  
 30 l'accesso è a word) l'indirizzo dell'accesso precedente;
- increment before: l'indirizzo base viene prima  
 incrementato di quattro e poi utilizzato per il  
 primo accesso. I registri successivi verranno  
 35 caricati in indirizzi ottenuti dal precedente

per incremento;

- decrement after: come increment after, ma l'indirizzo successivo è ottenuto per decremento
- decrement before: come increment before, ma gli indirizzi sono ottenuti per decremento.

5

Il registro base può opzionalmente essere aggiornato alla fine dell'operazione con il valore della successiva locazione puntata se il bit 21 (W bit) dell'opcode è a livello alto.

10

Esistono inoltre istruzioni di load & store multiple eseguibili solo in modo di funzionamento privilegiato che consentono di caricare dalla memoria il program counter o di accedere ai registri di uso generale del modo User.

15

Il processore ARM prevede poi altre due istruzioni che accedono alla memoria:

- SWP: swap word;
- SWPB: swap byte.

20

Queste istruzioni effettuano ciascuna due accessi a memoria, caricando in un primo registro il contenuto di una locazione di memoria puntata da un registro base e scrivendo nella stessa locazione di memoria il contenuto di un secondo registro. Se il primo e il secondo registro coincidono, si sono scambiati i contenuti del registro e della locazione di memoria.

25

Le operazioni sui coprocessori del modo 5 comprendono:

- load from memory to coprocessor;
- store from coprocessor to memory;
- move from general purpose register to coprocessor's register;
- move from coprocessor's register to general purpose register;
- execute coprocessor's data-processing operation.

30

35

Le istruzioni per i coprocessori non sono qui

descritte. Il processore ARM prevede poi tre istruzioni di salto:

- 5       - salto condizionato PC-relative (con e senza memoria dell'indirizzo di ritorno): l'offset di 24 bit è contenuto nell'opcode del salto. Per calcolare l'indirizzo di destinazione esso viene moltiplicato per quattro (in quanto ogni opcode del microprocessore ARM occupa 32 bit) ed esteso con segno, per poi essere sommato al valore attuale del program counter. È opportuno sottolineare che, in conseguenza dell'architettura della pipeline del processore ARM, al momento dell'aggiornamento che avviene nella fase di esecuzione, il program counter contiene l'indirizzo dell'istruzione di salto incrementato di otto;
- 10       - salto incondizionato con cambiamento di modo: il processore effettua un salto con offset di 24 bit, mantiene memoria dell'indirizzo di ritorno nel link register ed entra in modo Thumb, modificando il T bit della parola di stato;
- 15       - salto condizionato con cambiamento di modo (con o senza memoria dell'indirizzo di ritorno): il processore effettua un salto all'indirizzo contenuto in un registro indice. Il valore del registro indice viene allineato trascurandone il bit meno significativo, che viene utilizzato per decidere il modo di funzionamento (se a livello alto modo Thumb, altrimenti modo ARM).
- 20       - salto condizionato con cambiamento di modo (con o senza memoria dell'indirizzo di ritorno): il processore effettua un salto all'indirizzo contenuto in un registro indice. Il valore del registro indice viene allineato trascurandone il bit meno significativo, che viene utilizzato per decidere il modo di funzionamento (se a livello alto modo Thumb, altrimenti modo ARM).
- 25       - salto condizionato con cambiamento di modo (con o senza memoria dell'indirizzo di ritorno): il processore effettua un salto all'indirizzo contenuto in un registro indice. Il valore del registro indice viene allineato trascurandone il bit meno significativo, che viene utilizzato per decidere il modo di funzionamento (se a livello alto modo Thumb, altrimenti modo ARM).
- 30       Occorre sottolineare che, a differenza del processore LX, per il processore ARM il program counter fa parte dei registri di uso generale, quindi qualsiasi operazione di data-processing o di load dalla memoria che abbia R15 come registro destinazione può generare un salto.
- 35

La fase di commitment delle operazioni che hanno il program counter come destinazione è perciò diversa dalle normali istruzioni di load o data-processing e deve prevedere il ripristino del registro CPSR con il  
 5 valore contenuto nel registro SPSR associato al modo corrente.

Due istruzioni speciali si occupano della manipolazione dei registri di stato:

- MSR: muove un immediato o un registro di uso  
 10 generale del modo corrente in uno dei registri di stato del modo corrente (CPSR o SPSR);
- MRS: muove un registro di stato del modo corrente in un registro di uso generale del modo corrente.

15 Queste istruzioni sono eseguibili correttamente solo in un modo di esecuzione privilegiato e non devono essere usate per modificare il T bit del registro CPSR, il che causerebbe una transizione da modo ARM a modo Thumb o viceversa.

20 Accedere al registro SPSR nel modo System, che non vede questo registro, ha un effetto imprevedibile sull'esecuzione.

Viene ora descritta l'architettura del microprocessore LX.

25 Il processore LX è un nucleo o core con possibilità di assumere diverse configurazioni a seconda dell'impiego; nel seguito si farà riferimento alla versione single-cluster 4-issue.

L'architettura intera è a 32 bit e presenta 64  
 30 registri di uso generale più un program counter non accessibile direttamente dall'utente.

Due dei registri di uso generale hanno però funzioni particolari:

- il registro R63 è usato come link register;
- 35 - il registro R0 contiene sempre il valore zero ed

è utilizzato per confronti ed assegnamenti che non possono usare esplicitamente un'ulteriore campo immediato, come sarà chiarito nel seguito.

Esistono poi una serie di registri speciali (sempre  
5 a 32 bit) mappati in un'area riservata che occupa gli ultimi 4 Kbyte dello spazio di indirizzamento del processore LX, che è di 4 Gbyte.

Questi registri comprendono tra l'altro:

- 10 - un registro di stato PSW che contiene il modo di funzionamento (User o Supervisor) ed informazioni sui dispositivi per la protezione e la gestione della memoria;
- un registro di stack per il registro di stato, utilizzato in presenza di eccezioni;
- 15 - un registro di HANDLER\_PC, utilizzato in presenza di eccezioni per contenere l'indirizzo dell'exception-handler;
- altri registri che contengono informazioni funzionali al riconoscimento ed alla gestione  
20 delle eccezioni;
- registri per il controllo delle unità di protezione per la memoria programmi (IPU) e dati (DPU).

In ogni cluster del processore LX vi sono quindi  
25 quattro lane, ad ognuna delle quali è associata una ALU in grado di eseguire le normali operazioni-logico aritmetiche a 32 bit. Vi sono poi due unità in grado di effettuare le moltiplicazioni di un numero a 16 bit con uno a 32, con risultato troncato a 32 bit. Queste unità  
30 sono associate alle lanes 1 e 3 del cluster.

Il processore LX consente un solo accesso a memoria per ogni cluster, quindi esiste un'unica Load&Store unit, che può eseguire operazioni su word, halfword o byte e che può essere associata ad una qualsiasi delle  
35 lanes del cluster.

Un'unità denominata Instruction Issue Unit alloca le operazioni contenute in uno stesso bundle o insieme di istruzioni sulle lanes in modo tale che i due bit meno significativi dell'indirizzo di word di ogni istruzione determinino la lane sulla quale l'istruzione stessa è lanciata in esecuzione.

Una diretta conseguenza di questo è che un'istruzione di moltiplicazione, che deve essere eseguita su una lane dispari, deve occupare un'indirizzo di word dispari nella memoria programmi. Occorre quindi realizzare l'allineamento inserendo nel codice, se necessario, delle istruzioni NOP (no operation).

In ogni cluster è presente poi un'unità denominata branch unit che esegue le operazioni di salto. Il processore LX realizza le operazioni di salto condizionato sulla base di uno dei branch-bit register, un gruppo di otto registri da un bit ciascuno che contengono il risultato di operazioni logiche o di confronto.

Il valore di un branch-bit register deve essere assegnato almeno due bundle prima che avvenga il salto condizionato corrispondente.

Tutte le operazioni di salto devono occupare la prima istruzione del bundle e non possono esservi due istruzioni di salto all'interno dello stesso bundle, nemmeno se i due costrutti sono alternativi.

Il processore LX ha due sole modalità di indirizzamento per le istruzioni di data-processing:

- da registro;
- da immediato.

Gli immediati possono però essere di due tipi: short e long.

Gli immediati short sono numeri con segno su 9 bit, in grado di rappresentare un numero da -128 a +128 e

sono incorporati nei 32 bit dell'opcode.

5 Gli immediati long sono numeri con segno a 32 bit e occupano con i 9 bit meno significativi parte dei 32 bit dell'opcode. I rimanenti 23 bit sono contenuti in una delle word adiacenti all'opcode, con il vincolo di essere associati alle lane 0 o 2 del cluster e quindi di occupare un indirizzo di word pari.

10 Le operazioni di accesso alla memoria consentono solo l'indirizzamento mediante registro base più offset a 9 bit e, al contrario di quanto accade per il processore ARM, necessitano l'allineamento.

Accessi a word su indirizzi che non siano word-aligned, così come accessi ad halfword non halfword-aligned generano eccezioni.

15 Per quanto riguarda le istruzioni di salto, alle quali già si è accennato in precedenza, vi sono operazioni di salto condizionato (BR, BRF), che effettuano salti ad offset(23 bit) e istruzioni di salto incondizionato(CALL, GOTO, RTI) che possono  
20 effettuare salti ad offset(23 bit) oppure salti all'indirizzo puntato dal link-register, con il vincolo che il link-register deve essere modificato almeno tre bundle prima del salto corrispondente.

25 Vi sono poi due istruzioni (SLCT, SLCTF) che consentono di operare una operazione MOV condizionale sulla base della valutazione di un branch-bit: se questo ha livello alto viene portato nel registro destinazione il primo registro sorgente; in caso contrario viene caricato il secondo registro sorgente o  
30 un immediato a seconda della modalità di indirizzamento.

Occorre sottolineare infine che il processore LX, al contrario del processore ARM, non contiene un registro dei flag, e che quindi non è in grado di  
35 evidenziare automaticamente se le operazioni

aritmetiche generino carry o overflow.

Sono già note nella tecnica varie soluzioni che mirano a consentire ad un dato microprocessore di eseguire istruzioni di un set originariamente destinato ad un processore diverso.

Ad esempio, la domanda di brevetto europeo EP-A-0 747 808 descrive un processore a doppio set di istruzioni in grado di interpretare sia il codice nativo di un elaboratore IBM PowerPC sia il codice per la famiglia di processori x86 di Intel.

Detto documento descrive la gestione del sistema di memoria virtuale necessario a consentire il multitasking di due applicazioni sviluppate per set di istruzioni diversi, ma non descrive un processo di traduzione.

Per effettuare una traduzione efficiente delle istruzioni x86, la struttura originaria del PowerPC viene estesa con istruzioni e registri dedicati all'esecuzione in modalità x86.

La issue-logic del core viene inoltre modificata aggiungendo delle unità di decodifica e traduzione di opcode x86. Queste unità lavorano in parallelo all'unità di decodifica nativa del PowerPC ed in base al modo di lavoro corrente viene scelta quale delle due decodifiche applicare. Per consentire di determinare il modo di lavoro del processore viene aggiunta una Mode Control Unit responsabile di gestire la commutazione tra il modo x86 ed il modo PowerPC.

Quest'unità è in grado di interagire con la Memory Management Unit per consentire una corretta gestione del sistema di memoria virtuale.

#### Scopi e sintesi della presente invenzione

Analizzando i due set di istruzioni del processore ARM e LX emerge come solo una minima parte delle istruzioni del microprocessore ARM ha corrispondenza

con una singola istruzione del microprocessore LX, a causa della possibilità di esecuzione condizionale, della varietà dei modi di indirizzamento del processore ARM, delle diverse modalità di accesso a memoria e  
5 della mancanza su LX di un registro di stato. Una tale espansione del codice del processore ARM nella fase di traduzione ha un immediato riflesso sulla possibilità di emulare il comportamento di un processore ARM sul microprocessore LX e sulla possibile realizzazione di  
10 un dispositivo che effettua la traduzione.

La presente invenzione si prefigge pertanto lo scopo di fornire una soluzione che permetta di realizzare la traduzione delle istruzioni eseguibili su un processore di tipo ARM in istruzioni eseguibili su  
15 un processore di tipo LX.

Secondo la presente invenzione, tale scopo viene raggiunto grazie a un procedimento avente le caratteristiche richiamate in modo specifico nelle rivendicazioni che seguono, che formano parte  
20 integrante della presente descrizione.

L'invenzione riguarda anche il corrispondente dispositivo di traduzione, nonché il corrispondente prodotto informatico direttamente caricabile nella memoria di un elaboratore numerico quale un processore  
25 e comprendente porzioni di codice software per attuare il procedimento secondo l'invenzione quando il prodotto è eseguito su un elaboratore.

La soluzione secondo l'invenzione, messa a punto con specifico riferimento alla traduzione di istruzioni ARM in istruzioni LX, è in realtà applicabile ad un  
30 campo di impiego più ampio, ossia alla traduzione delle istruzioni di un microprocessore scalare pipelined avente caratteristiche comunque riconducibili alle caratteristiche di un processore ARM verso un  
35 microprocessore del tipo VLIW avente caratteristiche

comunque riconducibili alle caratteristiche di un processore LX.

Questo concetto è stato espresso nel seguito, ed in particolare nelle rivendicazioni annesse, facendo  
 5 riferimento rispettivamente a processori "di tipo ARM" e di "tipo LX".I

In sostanza, la soluzione secondo l'invenzione prevede di mappare ogni registro del microprocessore ARM su un registro del microprocessore LX atto a  
 10 emularne il comportamento, realizzando la traduzione in assenza di accesso diretto alle risorse del nucleo o core di detto microprocessore LX.

#### Breve descrizione dei disegni annessi

L'invenzione verrà ora descritta a puro titolo di  
 15 esempio, non limitativo, con riferimento ai disegni annessi, comprendenti un'unica figura che riproduce uno schema a blocchi di un dispositivo di traduzione operante secondo l'invenzione.

#### Descrizione di esempi preferiti di attuazione 20 dell'invenzione

I principi fondamentali della tecnica di traduzione qui descritta, corrispondente alla forma di attuazione dell'invenzione al momento preferita, sono i seguenti:

- mappare ogni registro del microprocessore ARM, compresi i registri replicati e tutti i registri di stato, su un registro del microprocessore LX che ne emulerà il comportamento;
- non modificare il core del microprocessore LX aggiungendo unità funzionali per coprire parte del set di istruzioni o delle modalità di indirizzamento del microprocessore ARM attualmente non coperte dal microprocessore LX;
- avere una traduzione univoca delle istruzioni del microprocessore ARM, che non sia data-dependent;

- non accedere mai direttamente alle risorse del core del microprocessore LX prima o durante la fase di traduzione.

5 In particolare, la soluzione secondo l'invenzione, nella sua forma di attuazione al momento preferita si distingue rispetto alle impostazioni note per diversi motivi:

10 - il core del microprocessore LX non viene in alcun modo modificato per interpretare il codice del microprocessore ARM, ma viene aggiunto un dispositivo traduttore esterno, posto tra detto core e la cache;

15 - il dispositivo traduttore, quando necessita di accedere alle risorse del core del microprocessore LX, non lo accede direttamente, ma incorpora nella traduzione dell'istruzione ARM dei costrutti condizionali basati sul contenuto dei registri o dei branch bit del core del microprocessore LX (si veda ad esempio nel seguito  
20 della descrizione il metodo con cui si determina il modo corrente di lavoro del processore ARM per le istruzioni MRS e MSR);

25 - il dispositivo traduttore entra in azione autonomamente, riconoscendo gli accessi alla zona di memoria riservata al codice ARM, senza bisogno di operazioni di commutazione o switch esplicite e di una Mode Control Unit;

30 - le istruzioni ARM vengono tradotte in istruzioni LX che vanno poi decodificate dalla issue-logic del microprocessore LX, la quale viene mantenuta inalterata. Al contrario, nel documento EP-A-707 848 sopracitato le istruzioni x86 vengono decodificate per controllare direttamente le risorse del core.

35 Mappando fisicamente tutti i registri del

processore ARM, compresi i registri replicati e i registri di stato, sui registri del processore LX, viene poi emulato anche il comportamento del program counter del processore ARM.

- 5 L'operazione di mappatura dei registri ARM e degli altri registri funzionali alla traduzione sui registri LX è descritta nella tabella 3 qui riprodotta.

R0: always zero	R32: ARM R13spv
R1: Rtemp 1 (temporary storage)	R33: ARM R14spv
R2: Rtemp 2 (temporary storage)	R34: ARM R13irq
R3: Rtemp3 (temporary storage)	R35: ARM R13irq
R4: Rtemp4 (temporary storage)	R36: ARM R13abt
R5: Rtemp5 (temporary storage)	R37: ARM R13abt
R6: Rtemp6 (temporary storage)	R38: ARM R13und
R7: Rtemp7 ((temporary storage)	R39: ARM R13und
R8: Rtemp8 (temporary storage)	R40: ARM R8stack
R9: Rt dest (temporary destination)	R41: ARM R9stack
R10: Rshift op (2nd operand)	R42: ARM R10stack
R11: RN (negative flag)	R43: ARM R11stack
R12: LX stack-pointer	R44: ARM R12stack
R13: RC (carry flag)	R45: ARM R13stack
R14: RV (overflow flag)	R46: ARM R14stack
R15: RZ (zero flag)	R47: not used
R16: ARM R0	R48: ARM CPSR (status register)
R17: ARM R1	R49: ARM SPSRspv (status register)
R18: ARM R2	R50: ARM SPSRirq (status register)
R19: ARM R3	R51: ARM SPSRfiq (status register)
R20: ARM R4	R52: ARM SPSRund (status register)
R21: ARM R5	R53: ARM SPSRabt (status register)
R22: ARM R6	R54: ARM R13fiq
R23: ARM R7	R55: ARM R13fiq
R24: ARM R8	R56: RtN (temporary negative flag)
R25: ARM R9	R57: RtZ (temporary zero flag)
R26: ARM R10	R58: RtC (temporary carry flag)

R27: ARM_R11	R59: RtV (temporary overflow flag)
R28: ARM_R12	R60: not used
R29: ARM_R13 (stack-pointer)	R61: not used
R30: ARM_R14 (link-register)	R62: not used
R31: ARM_R15/ ARM_PC	R63: LX link-register

**Tabella 3**

Con PC si indica il program counter del processore LX.

5 Per garantire la corretta esecuzione di un programma per microprocessore ARM su un microprocessore LX, una prima soluzione proposta è quella di forzare il program counter del microprocessore LX a emulare il funzionamento del program counter del microprocessore ARM.

10 Seguendo questo approccio, all'atto del caricamento di una istruzione ARM il program counter del processore LX deve contenere esattamente lo stesso valore del program counter del processore ARM, ma il processore LX si trova nella grande maggioranza dei casi a dover eseguire più di una istruzione per emulare il comportamento del processore ARM, cosicché al termine dell'esecuzione dell'istruzione emulata sarà necessario un salto all'indirizzo della prossima istruzione ARM, per poter caricare l'istruzione successiva.

20 Nel frattempo, un ulteriore registro ARM\_R15/ARM\_PC che emula il program counter del processore ARM, indicato in Tabella 3, dovrà essere prima incrementato di otto in modo che ogni istruzione che vi acceda durante la sua fase di esecuzione abbia un comportamento coerente con l'esecuzione sulla pipeline del processore ARM, e poi decrementato di quattro per puntare alla successiva istruzione ARM in memoria.

25 Fanno eccezione naturalmente le istruzioni ARM che hanno come registro destinazione il program counter, 30 per le quali il caricamento della successiva istruzione

ARM avverrà caricando nel link-register del processore LX il valore aggiornato del registro ARM\_PC ed effettuando un salto incondizionato del tipo GOTO link.

Per realizzare quanto descritto precedentemente, la  
 5 traduzione delle istruzioni ARM è affidata ad un dispositivo posto esternamente al core LX, che intercetta gli accessi alla zona di memoria riservata al codice ARM, traducendo tutte le istruzioni che non hanno una singola istruzione LX equivalente in un salto  
 10 incondizionato di tipo GOTO e forzando il program counter del processore LX a puntare ad un'area di memoria riservata per contenere la traduzione dell'istruzione ARM che nel frattempo viene decodificata.

15 In questo buffer di traduzione il dispositivo andrà a caricare tutti i bundle che costituiscono la traduzione LX dell'istruzione ARM decodificata. Come già detto, l'ultimo bundle della traduzione dovrà contenere un salto a link alla prossima istruzione ARM  
 20 da eseguire.

In questo modo, tutte le istruzioni ARM non direttamente mappabili su una istruzione LX necessitano di un salto alla zona di traduzione e di un salto a link per caricare la successiva istruzione ARM. Dato  
 25 che per realizzare un salto a link LX necessita che il link-register venga modificato almeno tre bundle prima del salto corrispondente, l'impatto di questi due salti sulla durata dell'esecuzione del programma ARM risulta notevole.

30 Per questo motivo è proposta una seconda soluzione, che prevede di non forzare il program counter del processore LX a seguire il funzionamento del program counter del processore ARM, in questo modo consentendo di risparmiare il salto alla zona di traduzione ed  
 35 effettuare il salto con link finale solo se

l'istruzione ARM realizza davvero un salto.

Oltre a rendere più veloce l'esecuzione del programma ARM questa seconda soluzione consente di risparmiare sul numero di istruzioni LX eseguite in  
 5 parallelo e diminuire di conseguenza la dissipazione di potenza da parte del core rispetto alla soluzione precedentemente proposta.

In assenza di salti, il program counter del processore LX viene lasciato evolvere liberamente.

10 Mentre nella prima soluzione se il program counter di LX esce dallo spazio di memoria nel quale risiedono le istruzioni ARM si è sicuri di eseguire istruzioni native LX, nella seconda soluzione il dispositivo traduttore deve intercettare tutti gli accessi del core  
 15 alla memoria istruzioni e controllare il registro puntatore per decidere se eseguire le prossime istruzioni come native LX o come ARM da emulare. Il registro puntatore risiede nel dispositivo traduttore mostrato in figura 1, che è posto tra instruction cache  
 20 e core del processore LX ed il cui funzionamento sarà descritto più in dettaglio nel seguito.

In figura 1 è mostrato dunque uno schema di principio del dispositivo di traduzione. Detto dispositivo di traduzione è indicato con 10 e comprende  
 25 un buffer di traduzione 11, un sottosistema di traduzione 12 associato a una tabella di microcodici e un'unità di controllo del dispositivo 13.

Il dispositivo di traduzione 10 è interposto fra un nucleo o core 14 del processore LX e una memoria cache  
 30 per le istruzioni 16, che esegue la propria funzione di caching su due aree di memoria, una memoria RAM 17 per le istruzioni ARM e una memoria RAM 18 per le istruzioni LX.

Il dispositivo di traduzione 10 dunque, posto  
 35 esternamente al core 14, intercetta gli accessi dei

detto core 14 alla memoria, in particolare alla zona di memoria riservata al codice ARM costituita dalla memoria RAM 17. Infatti detto dispositivo di traduzione 10 scambia dei dati per mezzo di detta unità di controllo del dispositivo 13 con il core o nucleo 14 del processore LX e riceve gli indirizzi su rispettive linee di segnale dati D1 e di segnale indirizzi A1.

L'unità di controllo del dispositivo 13 è connessa poi per mezzo di un bus indirizzi e dati AD alla memoria cache delle istruzioni 16, il cui puntatore è compreso in un insieme di registri di puntatori indicato con 15 e compreso nel dispositivo di traduzione 10.

Nella memoria 17 è possibile osservare un'istruzione ARM da tradurre, indicata con IA, a titolo di esempio l'istruzione BIC.

Si deve osservare a questo punto che, mentre nella prima soluzione, che prevede di forzare il program counter del processore LX, se detto program counter esce dallo spazio di memoria nel quale risiedono le istruzioni ARM si è sicuri di eseguire istruzioni native LX, nella seconda soluzione il dispositivo traduttore 10 deve intercettare tutti gli accessi del core alla memoria istruzioni e controllare il registro puntatore 15 per decidere se eseguire le prossime istruzioni come native LX o come ARM da emulare. Il registro puntatore 15, come sopra descritto, risiede nel dispositivo traduttore 10 che è posto tra la memoria cache 16 e core 14 del processore LX

All'atto del reset del processore LX di cui fa parte, il dispositivo traduttore 10 è inattivo e rimanda gli accessi a memoria ai dispositivi sottostanti, tipicamente alla instruction cache 16. Questa è la condizione in cui dispositivo traduttore 10 lavora finché il core 14 esegue un normale codice LX

(con istruzioni appartenenti all' instruction set del processore LX).

Quando si verifica un accesso alla zona di memoria riservata a contenere il codice ARM, la memoria 17, ad esempio un accesso all'istruzione IA, il dispositivo traduttore 10 si attiva, carica in un suo registro interno nel registro puntatori 15, indicato con NEXT\_ARM\_INSTR in fig. 1, l'indirizzo a cui si accede ed effettua la lettura dell'istruzione da tradurre IA in codice ARM dalla zona di memoria nella memoria 17 corrispondente. Il registro NEXT\_ARM\_INSTR ha cioè funzione di puntatore istruzioni ARM.

L'istruzione IA letta viene tradotta dal sottosistema di traduzione 12 che fa uso di una tabella di microcodice nel corrispondente insieme equivalente di istruzioni LX, indicato in figura 1 come traduzione T, e memorizzata nel buffer di traduzione 11. Il dispositivo di traduzione 10 alloca quindi "sopra" all'istruzione ARM una finestra di esecuzione alla quale saranno rimandati tutti gli accessi ad indirizzi di memoria che partono dal valore corrente del program counter del processore LX e coprono un'area pari a quella occupata dalla traduzione.

Il core 14 del processore LX può quindi leggere la prima istruzione della traduzione T dal buffer 11, il quale la invia all'unità di controllo 13, e il dispositivo 10 incrementa il registro NEXT\_ARM\_INSTR di quattro per puntare alla prossima istruzione ARM della traduzione. In presenza di salti nel programma in codice ARM il registro NEXT\_ARM\_INSTR deve essere riscritto esplicitamente mediante una operazione di store word contenuta nella traduzione LX dell'istruzione ARM.

Letta l'ultima istruzione della traduzione T, la finestra di esecuzione si chiude e, se al successivo

accesso in memoria il registro NEXT\_ARM\_INSTR punta al di fuori della zona di memoria riservata al codice ARM, il dispositivo di traduzione 10 si disattiva.

L'unità di controllo del dispositivo 13 controlla  
 5 l'attivazione del dispositivo di traduzione 10, attiva il sottosistema di traduzione nel blocco 12, per allocare e gestire la finestra di esecuzione mediante degli appositi registri puntatore interni che forniscono dei puntatori di finestra indicati con WP in  
 10 figura 1, e a propagare gli accessi a memoria dal core 14 verso il sistema di memoria costituito dalle memorie RAM 17 e 18.

Dell'insieme di istruzioni del processore ARM non potranno essere tradotte quelle istruzioni la cui  
 15 esecuzione dipende in modo diretto dall'hardware delle periferiche o del sistema di memoria del processore ARM specifico.

Queste istruzioni comprendono:

- software interrupt;
- 20 - breakpoint;
- istruzioni per la gestione dei coprocessori;
- accessi a memoria con l'opzione -T: queste operazioni interagiscono con la memoria realizzando un accesso in modalità User a prescindere  
 25 dall'effettivo modo corrente del processore. Il core ARM presenta infatti una linea che consente alla memoria di sapere con che attributi effettuare l'accesso (user o privileged).

Viene ora descritta ora in dettaglio la procedura  
 30 di traduzione delle istruzioni ARM in istruzioni LX.

A tal scopo verrà utilizzato uno pseudo-codice per descrivere le istruzioni.

La descrizione in pseudo-codice della traduzione di ogni istruzione ARM utilizza una sintassi C-like.

Le uniche estensioni alla tradizionale sintassi C sono le seguenti:

- essendo il processore LX una macchina VLIW, per evidenziare il parallelismo a livello di istruzione  
 5 ogni bundle, cioè l'insieme di istruzioni LX che possono essere eseguite in parallelo nello stesso ciclo, e' stato delimitato da una linea tratteggiata;
- tutte le operazioni di assegnamento all'interno di  
 10 un bundle devono considerarsi eseguite contemporaneamente: l'ordine con cui le istruzioni sono scritte all'interno di uno stesso bundle dello pseudo-codice non conta e non è quello realmente usato sul processore LX che impone dei vincoli nel  
 15 posizionamento di moltiplicazioni ed immediati lunghi (più di 9 bit);
- le variabili booleane memorizzate nei branch-bit register del processore LX sono indicate come \$< nome\_variabale >;
- 20 - gli immediati (costanti numeriche dipendenti dall'opcode della istruzione) generati direttamente dalla logica di traduzione in base al contenuto dell'opcode ARM sono indicati nel codice eseguito da LX come #< nome\_immediato >;
- 25 - l'elenco delle operazioni svolte dal dispositivo traduttore per generare questi valori è descritto nelle caselle con bordo tratteggiato poste alla destra della traduzione LX;
- tutti gli opcodes LX che fanno uso di immediati  
 30 lunghi occupano di fatto due word consecutive nel bundle LX. Nello pseudo-codice queste istruzioni non sono segnalate;
- l'operatore ASR (arithmetic shift right) simboleggia lo shift aritmetico a sinistra;

- l'operatore ROR (rotate right) simboleggia la rotazione verso destra;
- le macro 16lsb\_of (<register>) e 16msb\_of(<register>) indicano rispettivamente  
 5 l'estrazione dei 16 bit meno significativi e dei 16 più significativi da un registro. I restanti 16 bit sono riempiti con zeri;
- la macro Mask (<fields>) genera, sulla base della maschera a quattro bit <fields> contenuta  
 10 nell'opcode, le maschere per la modifica dei registri di stato per l'istruzione MSR;
- le operazioni di accesso a memoria sono descritte dalle macro: MemoryWord(<address>), MemoryByte(<address>), MemoryUByte (<address>),  
 15 MemorySByte(<address>). In particolare MemorySByte rappresenta la lettura dalla memoria all'indirizzo <address> di un byte che viene esteso con segno. MemoryUByte rappresenta la lettura di un byte unsigned;
- nella rappresentazione delle istruzioni, il  
 20 carattere @ e' usato come carattere jolly. Ad esempio ADD@ @ rappresenta tutte le istruzioni ARM di somma come ad esempio ADDEQ (somma se uguale) o ADDNE (somma se diverso), oppure ARM\_R@ rappresenta  
 25 uno qualsiasi dei registri LX che emulano i registri general-purpose del processore ARM, come ARM\_R1 o ARM\_R12.

Come già descritto in precedenza, il processore ARM consente l'esecuzione condizionata delle istruzioni  
 30 sulla base dei flag contenuti nel registro di stato CPSR.

La condizione è descritta nei quattro bit più significativi dell'opcode ARM.

Fanno eccezione l'istruzione BLX (branch, link and  
 35 exchange to Thumb state) e le istruzioni che fanno

riferimento ai coprocessori, che non sono condizionali.

La traduzione delle istruzioni per i coprocessori, inoltre, non è qui descritta in quanto per il loro corretto funzionamento è indispensabile la presenza nel  
 5 sistema basato sul core LX di dispositivi che emulino correttamente i coprocessori del processore ARM, qui non disponibili.

Anche dal punto di vista della traduzione si possono suddividere le istruzioni ARM nei cinque  
 10 gruppi:

- data-processing;
- moltiplicazioni;
- load&store singole;
- load&store multiple;
- 15 - salti.

Per tutte le cinque categorie la traduzione sul processore LX dell'operazione inizia con la verifica della condizione di esecuzione, che consiste nella valutazione di uno o più dei flag presenti nel registro  
 20 di stato. Questi flag sono quattro:

- N flag (negative flag): N=1 se il risultato di una operazione è negativo;
- C flag (carry flag): C è forzato a 1 se il risultato di una operazione logica o di una somma genera  
 25 riporto, a 0 se il risultato di una sottrazione genera riporto (borrow);
- V flag (overflow flag): V=1 se una operazione aritmetica ha generato overflow;
- Z flag (zero flag): Z=1 se il risultato di una  
 30 operazione è zero.

Le varie combinazioni di questi flag generano i sedici tipi di esecuzione condizionata AL, NV, EQ, NE, CS/HS, CC/LO, MI, PL, VS, VC, HI, LS, GE, LT, GT, LE.

Il processore LX, che non supporta l'esecuzione  
 35 condizionata, deve effettuare gli opportuni test per

valutare se la condizione di esecuzione è verificata, dopo di che può essere tradotta l'istruzione da eseguire.

La valutazione positiva determina l'impostazione di uno dei branch-bit del processore LX (in particolare il branch bit 7), che è utilizzato per una esecuzione predicata dell'istruzione o, nel caso di istruzioni che non possono essere eseguite in maniera speculativa come ad esempio le load&store, per saltare alla successiva istruzione ARM.

Per velocizzare la modifica dei flag in base al risultato delle operazioni di moltiplicazione e data-processing, ognuno dei quattro flag C, Z, N, V è calcolato e memorizzato in un apposito registro del processore LX. Detti quattro registri sono indicati con RC, RZ, RN, RV in Tabella 3.

Detto gruppo di registri RC, RZ, RN, RV viene letto per valutare la condizione di esecuzione delle successive istruzioni ARM. Il registro ARM\_CPSR sul processore LX che emula il comportamento del registro CPSR non viene quindi aggiornato in corrispondenza di ogni istruzione che lo modifica ma solamente quando si verifica un'operazione di lettura su di esso, in modo che il valore letto sia coerente con quello letto nella normale esecuzione dello stesso programma sul processore ARM.

Nella fase di traduzione della condizione viene inoltre incrementato di otto il valore del registro LX che emula il program-counter ARM, in modo che durante le fasi successive dell'operazione ogni accesso a tale registro veda il valore aggiornato, coerentemente con il comportamento del processore ARM.

La traduzione delle istruzioni ARM viene descritta in uno pseudo-codice, la cui sintassi è dettagliata più avanti.

Ad esempio, quando si incontra un' istruzione ARM con condizione di esecuzione GE (unsigned greater or equal), il primo bundle della traduzione LX sarà:

Istruzione ARM	Traduzione LX
@@@GE Rdest, Rsorg1, Rsorg2	\$Condition = (RN==RV)
	ARM_PC = ARM_PC + 8
	-----

Quando la condizione di esecuzione è ad esempio LE (signed less or equal) e non può essere valutata istantaneamente ma è necessario combinare i risultati dei due confronti  $Z=1$  e  $N!=V$ , saranno necessari due bundle per effettuare la valutazione.

Per le operazioni di data-processing, una volta tradotta la condizione occorre tradurre la modalità di indirizzamento.

Le modalità di indirizzamento del processore ARM per le istruzioni di data-processing, come descritto in precedenza sono nove:

15 immediato, diretto da registro, shift logico a sinistra da registro, shift logico a sinistra da immediato, shift logico a destra da registro, shift logico a destra da immediato, shift aritmetico a destra da registro, shift aritmetico a destra da immediato, rotazione a destra da registro, rotazione a destra da immediato, rotazione attraverso il carry flag.

In questa fase deve essere anche valutato l'ultimo bit in uscita dal registro sorgente in seguito all'operazione di shift. Questo bit sarà utilizzato per aggiornare il valore del carry flag per quelle operazioni logiche che richiedono il commitment e in questa fase viene memorizzato in un apposito registro temporaneo in attesa che in base all'opcode si decida se e come deve essere aggiornato il registro di stato CPSR.

Nel caso di indirizzamento da immediato, occorre

verificare se è necessario effettuare una rotazione sull'immediato ad otto bit contenuto nel byte meno significativo dell'opcode ARM. Il campo immediato dell'opcode ARM è lungo 12 bit e contiene nei quattro  
 5 bit più significativi un valore, indicato con amt, che descrive la rotazione a destra da applicare all'immediato ad 8 bit, indicato con imm.

Detto valore amt a quattro bit deve essere moltiplicato per due per sapere di quante posizioni  
 10 occorre ruotare verso destra l'immediato.

Se tale valore è uguale a zero non è necessaria la rotazione, ma occorre solamente spostare l'immediato in un apposito registro temporaneo, indicato con Rshift\_op in fig.1 in modo che la successiva traduzione della fase  
 15 di esecuzione dell'opcode sia univoca. Il valore del carry flag non viene alterato da questa modalità di indirizzamento, quindi nel registro di carry temporaneo, RtC in tabella 3, non si fa altro che scrivere il contenuto del registro di carry RC.

Istruzione ARM	Traduzione LX
@@@ Rdest.Rsorgl, imm	Condition Evaluation
	-----
	Rshift_op = #imm
	RtC = RC
	-----

20 Se invece è necessaria la rotazione, questa deve essere realizzata sul processore LX, che non ha nel suo set di istruzioni operazioni di rotazione, come una serie di shift ed or logici. Il valore del carry deve essere aggiornato con l'ultimo valore uscito dal  
 25 registro Rshift\_op in conseguenza della rotazione verso destra (che è anche il MSB dell'immediato ruotato).

L'indirizzamento diretto da registro è codificato nell'opcode ARM come caso particolare dell'indirizzamento con shift logico a sinistra da

immediato quando l'immediato è uguale a zero. La sua traduzione, analogamente all'indirizzamento da immediato, consiste solo nel muovere nel registro Rshift\_op il contenuto del registro sorgente, mentre il  
 5 carry non deve essere modificato.

Si analizzano ora i modi di indirizzamento da registro scalato con ammontare dello shift espresso da un immediato.

Shift logico a sinistra da immediato: l'entità  
 10 dello shift è espressa da un immediato a 5 bit contenuto nell'opcode. Oltre ad effettuare lo shift del registro sorgente, bisogna effettuare lo shift logico a destra complementare per aggiornare il carry. Con il solo shift il registro temporaneo di carry RtC non  
 15 sarebbe aggiornato correttamente, perchè il suo LSB conterrà il valore con cui aggiornare il flag, ma altri bit del registro possono avere contenuto non nullo. Sarà necessario, nelle fasi successive della traduzione, azzerare tutti gli altri bit del registro  
 20 RtC mediante un'operazione di AND logico.

Shift logico a destra da immediato: è analogo al caso precedente, ma il carry flag deve essere aggiornato con l'ultimo bit uscito dal registro Rshift\_op in seguito all'operazione di shift a destra.

25 Shift aritmetico a destra da immediato: è analogo al caso precedente, con la particolarità che se l'ammontare dello shift è zero si effettua uno shift aritmetico di 31 posizioni, cioè si mantiene nel registro il solo segno del contenuto del registro  
 30 Rsorg2.

Rotazione a destra da immediato: come già detto il processore LX non ha una istruzione di rotazione, quindi si deve realizzare la rotazione come serie di shift e OR logici.

35 Il carry flag va aggiornato con l'ultimo bit uscito

dal registro Rsorg2 in seguito alla rotazione a destra.

Shift logico a destra da registro: è analogo al caso precedente, ma se l'entità dello shift non è nulla il carry flag deve essere aggiornato con l'ultimo bit uscito da Rshift\_op in seguito all'operazione di shift a destra.

Shift aritmetico a destra da registro : è assolutamente analogo al caso precedente, con l'unica differenza che i due shift, ovviamente, devono essere aritmetici e non logici.

Rotazione a destra da registro: se il valore è nullo non deve dare alcun effetto. Altrimenti bisogna effettuare la rotazione ed aggiornare il carry flag con l'ultimo bit uscito dal registro Rsorg2 in seguito allo shift a destra.

Perchè la rotazione venga effettuata correttamente mediante gli shift, se l'ammontare della rotazione è maggiore o uguale a 31 occorre troncare (una rotazione a destra di 35 posizioni è infatti equivalente ad una rotazione di 3). Occorre comunque distinguere i casi di rotazione di zero dalla rotazione di multipli di 32, perchè nel secondo caso il carry flag deve essere aggiornato.

Una volta ottenuto l'operando, che viene sempre memorizzato nell'apposito registro Rshift\_op, viene tradotta la parte di esecuzione dell'opcode ARM.

Tra le istruzioni di data-processing vi sono 8 istruzioni logiche e 8 aritmetiche.

Le istruzioni logiche sono:

- 30 - AND (AND logico);
- EOR (OR esclusivo);
- ORR (OR inclusivo);
- BIC (bit clear);
- MOV (move to register);
- 35 - MVN (move negated);

- TST (test: aggiorna i flag in conseguenza di un AND logico);
- TEQ (test equivalence: aggiorna i flag in conseguenza di un OR esclusivo).

5        Queste istruzioni non modificano il flag di overflow e aggiornano il flag di carry in base alla precedente fase di predisposizione dell'operando, come descritto precedentemente.

10       Le operazioni di test, a differenza delle altre, non modificano il contenuto di nessun registro di uso generale ma solo i flag del registro CPSR.

Le istruzioni aritmetiche sono:

- ADD (somma);
- ADC (somma con riporto);
- 15 - SUB (sottrazione: sottrae il valore dello shifter-operand a quello di un registro;
- RSB (reverse subtract: sottrae il valore di un registro a quello dello shifter-operand);
- SBC (sottrazione con riporto borrow);
- 20 - RSC (reverse subtract con riporto borrow);
- CMP (compare: aggiorna i flag in conseguenza della sottrazione tra i due operandi);
- CMN (compare negated: aggiorna i flag in conseguenza della somma dei due operandi);

25       Queste istruzioni modificano il flag di overflow e il flag di carry in base al risultato.

La valutazione di questi due flag è diversa per la somma e per la sottrazione e viene effettuata nella successiva fase di commitment.

30       Le operazioni di confronto, a differenza delle altre, non modificano il contenuto di nessun registro di uso generale ma solo i flag del registro CPSR.

Visto che tutte le operazioni di data processing possono supportare l'esecuzione condizionale, il  
35 risultato dell'esecuzione viene memorizzato in un

registro destinazione temporaneo indicato come Rt\_dest in tabella 3.

- 5 Come esempio di istruzione logica del processore ARM, la parte di esecuzione dell'istruzione BIC (bit clear) è tradotta in questo modo:

Istruzione ARM	Traduzione LX
BIC @ @ Rdest, Rsorg1 @ @ @	Condition Evaluation
	-----
	2 <sup>nd</sup> Operand Generation
	-----
	Rtemp1 = Rsorg1 + Rshift_op
	-----
	Rt_dest = Rtemp1 + RC
	-----

- 10 Si noti come i bit alti del registro temporaneo di carry RtC vengano azzerati in questa fase della traduzione, dato che nella precedente fase di generazione del secondo operando tale registro può essere stato aggiornato semplicemente con un'operazione di shift.

Come esempio di operazione aritmetica, si considera l'istruzione ADC (add with carry):

Istruzione ARM	Traduzione LX
ADC @ @ Rdest, Rsorg1, @ @ @	Condition Evaluation
	-----
	2 <sup>nd</sup> Operand Generation
	-----
	Rtemp1 = Rsorg1 + Rshift_op
	-----
	Rt_dest = Rtemp1 + RC
	-----

- 15 Nel caso delle operazioni aritmetiche, qualora l'istruzione richieda l'aggiornamento dei flags, il carry flag andrà aggiornato in base al risultato della somma o della sottrazione e quindi non è necessario

mascherare il contenuto del registro RtC, che sarà ignorato nel seguito del processo di traduzione.

L'ultima fase della traduzione è quella di commitment. In questa fase, se la condizione di esecuzione è verificata, nel registro destinazione viene scritto il valore del registro destinazione temporaneo Rt\_dest.

Tutte le operazioni di data-processing possono poi opzionalmente modificare il registro di stato; fanno eccezione le operazioni di test e confronto che eseguono obbligatoriamente quest'operazione.

Dal momento che il processore LX non presenta un registro di stato, in questa fase verranno svolte, se richiesto dal valore alto del bit 21 dell'opcode ARM, una serie di istruzioni sul registro destinazione per stabilire come aggiornare i flag.

Il flag di zero viene portato a livello alto se il risultato di una operazione è zero.

Il flag di segno viene aggiornato con il bit più significativo del risultato.

Il flag di carry per le operazioni logiche viene aggiornato in base all'ultimo bit uscito dal registro sorgente in seguito allo shift nella fase di generazione degli operandi.

Per le operazioni di somma, C=1 indica la presenza di un riporto (carry) e viene impostato se:

- entrambi gli addendi sono negativi;
- uno degli addendi è negativo e il risultato è positivo.

Per le operazioni di sottrazione, C=0 indica la presenza di riporto (borrow). Considerando l'operazione  $RES = A - B$ , con o senza riporto, C=1 se:

- A è negativo e B positivo;
- uno dei termini è negativo e il risultato positivo.

Il flag di overflow non è modificato dalle

operazioni logiche.

Per le operazioni di somma,  $V=1$  se:

- entrambi gli operandi sono positivi e il risultato negativo;
- 5 - entrambi gli operandi sono negativi ed il risultato positivo;

Per le operazioni di sottrazione (vedi sopra),  $V=1$  se:

- 10 - A è negativo, B è positivo e il risultato è positivo;
- A è positivo, B è negativo e il risultato è negativo.

È per velocizzare questa fase che ognuno dei quattro flag (C, Z, N, V) è calcolato e memorizzato in un apposito registro temporaneo (RtC, RtZ, RtN, RtV) per poi essere salvato, se la condizione di esecuzione è verificata, in un registro opportuno (RC, RZ, RN, RV).

Questo ultimo gruppo di registri sarà poi usato per valutare la condizione di esecuzione delle successive istruzioni ARM. Si sottolinea ancora che il registro ARM\_CPSR che emula il comportamento del registro CPSR del processore ARM non viene aggiornato in corrispondenza di ogni istruzione che lo modifica ma solamente quando si verifica un'operazione di lettura su di esso, in modo che il valore letto sia coerente con quello letto nella normale esecuzione dello stesso programma su ARM.

Oltre alla modifica dei flag, in questa fase viene aggiornato il valore del registro del processore LX che emula il program counter del processore ARM indicato con ARM\_R15/ARM\_PC in tabella 3.

Se R15 non è il registro destinazione, al valore corrente (ovvero  $PC + 8$ ) viene sottratto 4 per puntare alla prossima istruzione ARM.

Se R15 è il registro destinazione, il registro ARM\_PC viene aggiornato di conseguenza ed il processore LX è forzato ad effettuare un salto a link al nuovo valore di tale registro.

5        Se inoltre occorre aggiornare il registro di stato, nel registro CPSR viene caricato il registro SPSR associato al modo corrente ed occorre eseguire una complessa procedura di switch del modo di lavoro del processore ARM .

10       Se l'istruzione non causa un salto, il dispositivo traduttore è in grado autonomamente di puntare alla successiva istruzione ARM in memoria, mentre se l'operazione modifica il contenuto del program-counter del processore ARM è necessario forzare il dispositivo  
15 a puntare al nuovo valore di ARM\_PC.

Questo è facilmente ottenibile scrivendo il nuovo valore del program-counter ARM in memoria all'indirizzo associato all'Istruzione ARM-pointer del dispositivo traduttore.

20       Per le istruzioni aritmetiche di somma (ADD,ADC) che richiedono l'aggiornamento del registro di stato e non causano salti, si ricorda che l'aggiornamento dei flag di carry ed overflow avviene in questo modo:

25       C=1 indica la presenza di un riporto (carry) e viene impostato se:

- entrambi gli addendi sono negativi
- uno degli addendi è negativo e il risultato è positivo

V=1 se:

- 30       - entrambi gli operandi sono positivi e il risultato negativo
- entrambi gli operandi sono negativi ed il risultato positivo

35       La fase di commitment è quindi tradotta sul processore LX in questo modo:

## Istruzione ARM

ADDS@@ Rdest, Rsorgl, @@@

## Traduzione LX

Condition Evaluation

-----

2<sup>nd</sup> Operand Generation

-----

Execution

-----

RtN=Rt\_dest»31

Rtemp1 = Rshift\_op » 31

Rtemp2 = Rsorgl » 31

Rdest = (\$Condition)?

Rt\_dest : Rdest

-----

Rtemp3 = Rtemp1 && Rtemp2

Rtemp4 = (Rtemp1 == 0) &&  
(Rtemp2 == 1)

Rtemp1 = Rtemp1 && (RtN ==  
0)

Rtemp2 = Rtemp2 && (RtN ==  
0)

-----

Rtemp1= Rtemp1 || Rtemp2

Rtemp2 = Rtemp4 && (RtN ==  
0)

Rtemp4 = Rtemp3 && (RtN ==  
0)

RN = (\$Condition)? RtN : RN

-----

RtV = Rtemp2 [| Rtemp4

RtC = Rtemp1 [| Rtemp3

RtZ = (Rt\_dest == 0)

-----

RC = (\$Condition)? RtC : RC

RV = (\$Condition)? RtV : RV

RZ = (\$Condition)? RtZ : RZ

ARM PC = ARM PC - 4

Per l'istruzione di confronto CMN, che effettua una somma ma non va a scrivere il risultato dell'operazione in un registro di uso generale, la traduzione della fase di commitment diventa ovviamente la seguente:

Istruzione ARM  
CMN@@Rsorgl,@@@

Traduzione LX  
Condition Evaluation  
-----  
2<sup>nd</sup> Operand Generation  
-----  
Execution  
-----  
RtN=Rt\_dest»31  
Rtemp1 = Rshift\_op » 31  
Rtemp2 = Rsorgl » 31  
-----  
Rtemp3 = Rtemp1 && Rtemp2  
Rtemp4 = (Rtemp1 == 0) &&  
(Rtemp2 == 1)  
Rtemp1 = Rtemp1 && (RtN ==  
0)  
Rtemp2 = Rtemp2 && (RtN ==  
0)  
-----  
Rtemp1 = Rtemp1 || Rtemp2  
Rtemp2 = Rtemp4 && (RtN ==  
0)  
Rtemp4 = Rtemp3 && (RtN ==  
0)  
RN = (\$Condition)? RtN : RN  
-----  
RtV = Rtemp2 || Rtemp4  
RtC = Rtemp1 || Rtemp3  
RtZ = (Rt\_dest == 0)  
-----

RC = (\$Condition)? RtC : RC

RV = (\$Condition)? RtV : RV

RZ = (\$Condition)? RtZ : RZ

ARM PC = ARM PC - 4

Per le istruzioni aritmetiche di sottrazione (SUB, SBC, RSB, RSC) che richiedono l'aggiornamento del registro di stato e non causano salti, si ricorda che l'aggiornamento dei flag di carry ed overflow avviene

5 in questo modo:

C=0 indica la presenza di riporto (borrow). Considerando l'operazione  $RES = A - B$ , con o senza riporto, C=1 se:

- A è negativo e B positivo;

10 - uno dei termini è negativo e il risultato positivo.

V=1 se:

- A è negativo, B è positivo e il risultato è positivo;

- A è positivo, B è negativo e il risultato è  
15 negativo.

Quando invece il registro destinazione è il program-counter del processore ARM ed è richiesto l'aggiornamento del registro di stato, è necessario leggere i 5 LSB del registro di stato per individuare  
20 il modo di lavoro corrente e scegliere quale delle repliche del registro di stato SPSR salvare nel registro CPSR. Segue una complessa procedura che consente di effettuare lo switch del modo di lavoro corrente del processore ARM, che di fatto ricalca quasi  
25 perfettamente la traduzione dell'istruzione ARM MSR (move to status register from general-purpose register).

Per la descrizione dettagliata di questa procedura si rimanda perciò alla successiva descrizione della  
30 traduzione dell'istruzione MSR.

Ad ogni modo la traduzione dell'istruzione ADDS che

usa come registro destinazione il program counter predispone il link-register di LX, detto LX\_LR, ad effettuare il salto e forza il dispositivo traduttore a puntare al nuovo valore di ARM\_PC.

- 5       Questo è ottenuto scrivendo il nuovo valore del program-counter ARM in memoria all'indirizzo associato all'Istruzione ARM-pointer del dispositivo traduttore.

Analogamente, se non è richiesto l'aggiornamento del registro di stato, CPSR non deve essere modificato  
10 e la traduzione si semplifica.

Ovviamente l'uso di immediati a 32 bit nella traduzione (ad esempio ARMPINTER\_ADDR) deve seguire i vincoli imposti dal processore LX per la codifica degli immediati. Il dispositivo traduttore deve assicurare  
15 che gli immediati lunghi vengano posizionati in indirizzi di memoria corrispondenti ad un numero di word pari, come richiesto dal core LX.

I 3 LSB dell'indirizzo in cui si codifica l'immediato lungo devono essere cioè sempre nulli.

- 20       Similarmente alle istruzioni di data-processing si comportano le istruzioni di moltiplicazione e moltiplicazione con accumulo. Queste istruzioni supportano però solamente l'indirizzamento diretto da registro e, sebbene supportino l'opzione -S, non  
25 modificano mai i flag di carry e di overflow ma solo quelli di segno e di zero.

Mentre il processore ARM è in grado di effettuare anche moltiplicazioni di numeri a 32 bit ottenendo un risultato a 64 bit che viene spezzato su due registri  
30 destinazione, il processore LX può solamente effettuare moltiplicazioni di numeri a 16 bit o di un numero a 16 bit con uno a 32, troncando comunque il risultato a 32 bit.

- 35       Per questo motivo le moltiplicazioni 32x32 del processore ARM devono essere realizzate sul processore

LX come una serie di moltiplicazioni 16x16 e di somme con riporto, seguendo il seguente procedimento.

Siano A e B i due operandi a 32 bit contenuti nei due registri sorgente; si indicano con AH e BH le halfword alte di A e di B e con AL e BL le halfword basse.

$$A = AL + AH * 2^{16}$$

$$B = BL + BH * 2^{16}$$

$$A * B = AL * BL + (AH * BL + AL * BH) * 2^{16} + AH * BH * 2^{32}$$

Nel caso di moltiplicazione 32x32 di tipo signed, prima si determinano i valori assoluti di A e calcolando il complemento a due dei numeri negativi, dopo di che si effettua la moltiplicazione unsigned come descritto in precedenza. In base al segno degli operandi già estratto in precedenza si calcola quindi il segno del risultato, che se negativo richiede di effettuare il complemento a due del numero a 64 bit ottenuto dalla moltiplicazione dei valori assoluti.

Una volta valutata la condizione di esecuzione, viene tradotta l'esecuzione della moltiplicazione.

Se l'istruzione è una MUL, si effettua una moltiplicazione di due numeri a 32 bit con risultato troncato a 32 bit. Il posizionamento nei bundle delle operazioni di moltiplicazione deve rispettare i vincoli imposti dal processore LX. Il dispositivo traduttore deve assicurare che le moltiplicazioni vengano posizionate in indirizzi di memoria corrispondenti ad un numero di word dispari, come richiesto dal core LX.

I 3 LSB dell'indirizzo in cui si codifica l'immediato lungo devono cioè essere 1 0 0.

Devono inoltre essere rispettati i vincoli sulla latenza delle moltiplicazioni, che è doppia rispetto a quella delle istruzioni di data-processing.

Se l'istruzione è una MLA (multiply and

accumulate), al risultato di una istruzione MUL deve essere aggiunto il contenuto di un terzo registro sorgente Rsorg3.

Se l'istruzione è una UMULL (unsigned multiply), si deve effettuare la moltiplicazione di due numeri unsigned a 32 bit e spezzare il risultato a 64 bit ottenuto su due registri: la parte alta è salvata nel registro temporaneo Rt\_dest in attesa della fase di commitment, mentre la parte bassa è salvata nel registro temporaneo Rtemp1.

Per l'istruzione UMLAL (unsigned multiply and accumulate), occorre soltanto sommare al risultato della istruzione UMULL il numero a 64 bit precedentemente contenuto nei registri RdHi e dLo, ricordandosi di propagare il riporto della somma.

Se l'istruzione è una SMULL (signed multiply), si deve effettuare la moltiplicazione di due numeri signed a 32 bit in complemento a due e spezzare il risultato a 64 bit ottenuto su due registri: la parte alta è salvata nel registro temporaneo Rt\_dest in attesa della fase di commitment, mentre la parte bassa è salvata in Rtemp1.

Per effettuare la moltiplicazione signed, prima si calcolano i valori assoluti dei due operandi, poi si effettua la moltiplicazione unsigned come per l'istruzione UMULL ed alla fine, se i due operandi avevano segno discorde, si effettua il complemento a due del numero a 64 bit risultante dalla moltiplicazione dei valori assoluti.

La traduzione è quindi la seguente:

Istruzione ARM	Traduzione LX
SMULL@@ RdHi,RdLo,Rsorg1,	Condition Evaluation
Rsorg2	-----
	Rtemp1 = - Rsorg1
	Rtemp2 = - Rsorg2

```

-----
Rtemp3=Rsorg1»31
Rtemp4 = Rsorg2 » 31
-----
$Neg1 = (Rtemp3 == 1)
$Neg2 = (Rtemp4 == 1)
$PosRes = (Rtemp3 == Rtemp4)
-----
Rtemp7 = ($Neg1)? Rtemp1 :
Rsorg1
Rtemp8 = ($Neg2)? Rtemp2 :
Rsorg2
-----
$Carry0 = 0
Rtemp1 = 16lsb_of (Rtemp7) *
16msb_of(Rtemp8)
Rtemp2 = 16lsb_of (Rtemp8) *
16msb_of(Rtemp7)
-----
Rtemp3 = 16lsb_of(Rtemp7) *
16lsb_of(Rtemp8)
Rtemp4 = 16msb_of(Rtemp7) *
16msb_of(Rtemp8)
-----
Rtemp1 = Rtemp1 « 16
Rtemp5 = Rtemp1 » 16
Rtemp2 = Rtemp2 « 16
Rtemp6 = Rtemp2 » 16
-----
Rtemp5 = Rtemp5 + Rtemp6
$Carry1,Rtemp1 = Rtemp1 +
Rtemp3 + $Carry0
-----
$Carry2,Rtemp8 = Rtemp4 +
Rtemp5 + $Carry1

```

```

$Carry1,Rtemp7 = Rtemp1 +
Rtemp2 + $Carry0
-----
$Carry2,Rtemp8 = Rtemp8 +
$Carry1
$LowIsZero = (Rtemp7 == 0)
Rtemp1 = - Rtemp7
-----
Rtemp3 = ~ Rtemp8
Rtemp1 = ($PosRes)? Rtemp7 :
Rtemp1
-----
$Carry2,Rtemp2 = Rtemp3 +
$LowIsZero
-----
Rt_dest = ($PosRes)? Rtemp8
: Rtemp2
-----

```

Si noti che, per effettuare il complemento a due del risultato a 64 bit occorre verificare se i 32 bit meno significativi siano tutti zeri o meno per decidere se la parte superiore debba essere complementata a due o negata.

5

Per l'istruzione SMLAL (signed multiply and accumulate), occorre soltanto sommare al risultato dell'istruzione SMULL il numero a 64 bit precedentemente contenuto nei registri RdHi e RdLo, ricordandosi di propagare il riporto della somma.

10

Come le istruzioni di data-processing, le moltiplicazioni supportano l'opzione -S e terminano con una fase di commitment analoga che però non aggiorna i flag di overflow e carry.

15

Dal punto di vista della traduzione, le operazioni di accesso a memoria differiscono fondamentalmente dalle operazioni di data-processing perchè non possono

essere eseguite in maniera predicativa. Infatti, mentre è possibile effettuare comunque una somma o una moltiplicazione per poi decidere se scriverne o meno il risultato nel registro destinazione, questo approccio  
 5 evidentemente non ha senso per le operazioni di scrittura in memoria.

Tale metodo inoltre non è applicabile nemmeno per le operazioni di lettura, perchè se l'accesso in lettura avviene in una locazione di memoria sulla quale  
 10 è mappato un dispositivo (si pensi ad esempio ad una UART), l'accesso può causare la perdita o la modifica dell'informazione contenuta in quella locazione.

Anche per i motivi espressi sopra, la traduzione delle operazioni sulla memoria segue un procedimento  
 15 differente da quello per le istruzioni di data-processing.

Le istruzioni di accesso singolo in memoria comprendono le istruzioni del modo 2 che quelle del modo 3. Tutte queste istruzioni impiegano per  
 20 l'indirizzamento un registro base al quale viene sommato o sottratto un offset che può essere ottenuto con varie modalità.

Le istruzioni del modo 2 sono le operazioni di load&store di word e unsigned byte. Le modalità di  
 25 indirizzamento supportate sono nove:

- registro base +/- immediato a 12 bit;
- registro base +/- registro offset;
- registro base +/- registro offset scalato (il registro offset è shiftato con modalità analoghe  
 30 alle istruzioni di data-processing; l'entità dello shift è descritta da un immediato);
- registro base +/- immediato pre-indexed (il registro base viene aggiornato prima di effettuare l'accesso a memoria);
- 35 - registro base +/- registro offset pre-indexed;

- registro base +/- registro scalato pre-indexed;
- registro base +/- immediato post-indexed (il registro base viene aggiornato dopo avere effettuato l'accesso a memoria);
- 5 - registro base +/- registro offset post-indexed;
- registro base +/- registro scalato post-indexed.

Le operazioni di load & store in memoria del modo 3 agiscono su halfword e signed byte. Le modalità di indirizzamento supportate sono sei delle nove associate al modo 2:

- registro base +/- immediato a 8 bit;
- registro base +/- registro offset;
- registro base +/- immediato pre-indexed;
- registro base +/- registro offset pre-indexed;
- 15 - registro base +/- immediato post-indexed;
- registro base +/- registro offset post-indexed.

Come spiegato in precedenza, il processore ARM consente di realizzare accessi a word con indirizzi non word-aligned, mentre il processore LX non lo consente ed in questi casi scatena un'eccezione. Per questo motivo la traduzione degli accessi in memoria del processore ARM deve necessariamente prevedere una operazione di troncamento per generare indirizzi word o halfword-aligned. Inoltre essendo sia il processore ARM che LX processori potenzialmente bi-endian, qualora l'endianness, cioè la rappresentazione degli interi da destra a sinistra o sinistra a destra, dei due sistemi sia diversa è necessario effettuare l'opportuna operazione di swap per word o halfword.

La traduzione delle istruzioni dei modi 2 e 3 avviene con due processi analoghi, che si differenziano solo per i modi di indirizzamento supportati e per le diverse traduzioni della fase di accesso a memoria. Queste differenze, come si vedrà meglio in seguito, sono finalizzate ad ottenere in ogni caso la traduzione

più velocemente eseguibile su processore LX .

Dopo la prima fase di decodifica e traduzione del campo condizionale come per le istruzioni di data-processing, si ha la traduzione del modo di indirizzamento e la conseguente generazione dell'indirizzo per l'accesso a memoria.

In questa fase si calcolano sia l'indirizzo per l'accesso in memoria sia il valore con cui, se necessario, si dovrà aggiornare il registro base al termine dell'esecuzione dell'istruzione.

I due valori vengono memorizzati rispettivamente nei registri temporanei Rshift\_op e Rtemp6.

Viene quindi tradotta l'operazione di load o store, che come spiegato in precedenza deve per prima cosa verificare se la condizione di esecuzione è verificata o meno e, in caso negativo, non effettuare l'accesso ma saltare all'ultimo bundle della traduzione per poi effettuare il salto a link ed accedere quindi alla successiva istruzione ARM da eseguire.

Si analizza ora nel dettaglio la traduzione di alcuni dei vari modi di indirizzamento, partendo da quelle comuni ai modi 2 e 3.

La modalità di indirizzamento è descritta dai bit che vanno dal 21 al 25 dell'opcode ARM.

Nel caso di indirizzamento con registro base +/- immediato, il registro base non deve essere aggiornato, mentre il registro Rshift\_op, che sarà utilizzato come puntatore alla memoria, deve contenere il risultato della somma (o della sottrazione) del registro base e dell'immediato lungo offset:

Istruzione ARM	Traduzione LX
LDR@ @ Rdest, [Rbase, # +1- imm]	Condition Evaluation
	-----
	Rtemp6 = Rbase
	Rshift_op = Rbase +

#sign\_imm sign\_imm = +/-

sign\_imm = +/- imm

Nel caso di indirizzamento con immediato pre-indexed, anche il registro base deve essere aggiornato alla fine dell'operazione, quindi si va a modificare il registro Rtemp6 di conseguenza.

- 5     Nel caso di indirizzamento con immediato post-indexed, il registro base dovrà essere aggiornato alla fine dell'operazione ma l'accesso a memoria dovrà essere effettuato al valore corrente del registro base.

- 10    Nel caso di indirizzamento con offset contenuto in un registro, si opera analogamente.

- Le operazioni del modo 2 supportano anche gli indirizzamenti con registro offset scalato (con le solite modalità LSL, LSR, ASR, ROR, RRX). La traduzione di queste modalità consiste dapprima nell'ottenimento  
15    dell'offset mediante le operazioni di shift o rotazione, e poi nell'aggiornamento dei registri Rtemp6 e Rshift\_op con le modalità già viste.

- Una volta tradotto il modo di indirizzamento viene tradotta l'esecuzione dell'accesso a memoria. La  
20    traduzione dell'accesso varia totalmente in base al formato del dato da leggere o scrivere in memoria.

- Gli accessi a byte non presentano problemi di endianness nè di allineamento. Esempi di accessi a byte sono l'istruzione STRB (store unsigned byte, modo 2),  
25    l'istruzione LDRB (load unsigned byte, modo 2).

- Si noti che, a causa della latenza dell'operazione di load byte che richiede di attendere due bundle per accedere al byte letto, si deve inserire un bundle vuoto al fine di rendere il contenuto del registro  
30    Rdest immediatamente disponibile all'istruzione ARM successiva ed evitare così read-hazards.

Si noti inoltre che, essendo possibile che il registro destinazione coincida con il registro base

quando quest' ultimo non richiede di essere aggiornato dalle modalità post- o pre-indexed, occorre realizzare in due bundle separati prima la scrittura nel registro Rbase e poi quella nel registro Rdest.

- 5       L'istruzione LDRSB (load signed byte) fa parte del modo 3 e come tale non ammette l'indirizzamento con registro offset scalato. Il byte letto è esteso con segno.

- 10       Il processore LX richiede che le operazioni di memoria che coinvolgono halfword siano halfword-aligned, mentre il processore ARM all'atto dell'accesso in memoria ignora l'ultimo bit dell'indirizzo, per poi eventualmente scambiare i due byte letti tra di loro se il bit scartato era un 1.

- 15       Le operazioni di store invece ignorano completamente l'ultimo bit.

- 20       Occorre inoltre prestare attenzione alle endianness del processore ARM e LX: se i due sistemi usano convenzioni diverse è necessario scambiare tra di loro i byte dell'halfword letta.

- 25       L'accesso ad halfword del processore ARM sarà quindi tradotto in due singoli accessi a byte di tipo unsigned per rendere più veloce l'esecuzione nel caso in cui è necessario uno swap dei byte della halfword (cioè trasformare la parola esadecimale xxxxYyZz nella parola xxxxZzYy).

- 30       L'istruzione LDRH (load unsigned halfword, modo 3) legge byte per byte la halfword dal registro destinazione e la scrive nel registro destinazione effettuando o meno lo swap in base all'endianness del processore ARM ed al bit meno significativo dell'indirizzo. L'altra halfword del registro destinazione viene riempita di zeri.

- 35       Si sottolinea come, per risparmiare tempo nel caso in cui fosse necessario effettuare lo swap dei byte

letti, vengono precalcolate sia la word letta che la sua versione swapped e poi si sceglie la versione corretta in base anche all'endianness del sistema ARM.

5 L'endianness del processore LX in questo caso è influente perchè gli accessi sono realizzati byte per byte.

L'istruzione LDRSH (load signed halfword, modo 3) è simile alla precedente con l'unica differenza che i byte vengono letti con segno e poi il meno significativo è mascherato con 0x00FF per costruire la halfword già sign-extended.

La strategia di traduzione, per il resto, è identica al caso precedente.

15 Nel caso big endian, è ancora sufficiente scambiare i ruoli di Rtemp1 ed Rtemp2 nell'ultimo bundle.

Per la traduzione degli accessi a word è scelto un'approccio differente: rispetto ad effettuare quattro accessi a byte consecutivi per poi costruire la word in base all'endianness del processore ARM, si preferisce effettuare un singolo accesso a word allineato per poi eventualmente effettuare lo swap dei byte qualora le endianness del processore ARM e del processore LX siano diverse.

25 In tal modo, se il processore ARM ed il processore LX hanno la stessa endianness il guadagno in termini di velocità di esecuzione è notevole, mentre quando le endianness sono diverse, le due soluzioni, con quattro accessi o con singolo accesso, sono pressoché equivalenti.

30 L'istruzione STR (store word, modo 2) ignora i due bit meno significativi dell'indirizzo ed è quindi self-aligned. Quando le endianness del processore ARM e di LX sono diversi, la traduzione dell'istruzione comprende lo swap dei byte della word letta: la parola  
35 esadecimale 0xAaBbCcDd viene trasformata in 0xDdCcBbAa.

Se le due endianness invece sono uguali tutti i bundle dedicati allo swap sono risparmiati.

L'istruzione LDR (load word, modo 2) ha un processo di traduzione più complesso per vari motivi:

- 5 - l'accesso è effettuato trascurando i due bit meno significativi dell'indirizzo, ma poi la parola letta deve essere swapped se le endianness del processore ARM e del processore LX sono diverse ed anche
- 10 ruotata verso destra di un numero di posizioni pari ad otto volte il valore dei due bit di indirizzo ignorati (ad esempio, se l'indirizzo termina per 11 la word deve essere ruotata di 24 posizioni);
- se il registro destinazione della operazione load è il program-counter del processore ARM, l'istruzione
- 15 può generare un salto ed occorre quindi assicurarsi che il valore caricato sia word-aligned e predisporre il dispositivo traduttore a puntare all'istruzione ARM obiettivo del salto.

20 A prescindere dal fatto che il registro destinazione sia o meno il registro ARM\_PC, la prima parte della traduzione è la stessa e, nel caso in cui sia necessario effettuare lo swap, si ha:

Istruzione ARM	Traduzione LX
LDR@@ Rdest, [ Rbase, @ @ @	Condition Evaluation
	-----
	Address generation
	-----
	Rshift_op = Rshift_op &
	OxFFFFFFFFC
	Rtemp5 = Rshift_op & 0x03
	Rbase = (\$Condition)? Rtemp6
	: Rbase
	IF (! \$Condition) GOTO end
	-----
	Rt_dest =

		MemoryWord(Rshift_op)
		Rtemp5 = Rtemp5 « 3
Bundle	dedicati	Rtemp1 = 0x00FF
esclusivamente		-----
All'operazione di swap		Rtemp6 = 32 - Rtemp5
		-----
		Rtemp1 = Rtemp1 « 8
		Rtemp3 = Rtemp1 « 16
		Rtemp2 = Rt_dest » 8
		Rtemp4 = Rt_dest « 8
		-----
		Rtemp1 = Rt_dest » 24
		Rtemp3 = Rt_dest « 24
		Rtemp2 = Rtemp2 & Rtemp1
		Rtemp4 = Rtemp4 & Rtemp3
		-----
		Rtemp2 = Rtemp1   Rtemp2
		Rtemp4 = Rtemp4   Rtemp3
		-----
		Rt_dest = Rtemp2   Rtemp4
		-----
		Rt_dest = Rt_dest » Rtemp5
		Rtemp1 = Rt_dest « Rtemp6
		-----
		Rdest = Rtemp1   Rt_dest

Effettuato l'accesso alla memoria e completate le eventuali operazioni di swap e rotazione della parola letta, se il registro destinazione non è ARM\_PC la traduzione termina semplicemente aggiornando il program counter stesso:

Istruzione ARM	Traduzione LX
LDR@ @ Rdest, [ Rbase, @ @ @	Condition Evaluation
	-----
	Address generation
	-----

## Memory Access

-----  
 end: ARM\_PC = ARM\_PC - 4

Quando invece il registro destinazione è proprio ARM\_PC, la versione 5 dell'instruction-set del processore ARM richiede che la parola letta sia resa halfword-aligned e che il bit meno significativo della parola stabilisca se entrare o meno in Thumb-state, sellando il bit 5 del registro di stato ARM\_CPSR.

Occorre poi predisporre il dispositivo a puntare all'indirizzo destinazione del salto.

Istruzione ARM

LDR@@ R15, [Rbase, @@@

Traduzione LX

Condition Evaluation

-----  
 Address generation  
 -----

Memory Access  
 -----

Rtemp1 = ARM\_PC & 1 \_\_\_\_\_  
 LXJLR = ARM\_PC & 0xFFFFFFFF  
 Rtemp3 = ARMPINTER\_ADDR

-----  
 Rtemp1 = Rtemp1 « 5 \_\_\_\_\_  
 ARM.CPSR = ARM\_CPSR &  
 0xFFFFFDF  
 ARM\_PC = ARM\_PC & 0xFFFFFFF

-----  
 MemoryWord (Rtemp3) = ARM\_PC  
 ARM\_PC = ARM\_PC + 4  
 ARM.CPSR = ARM\_CPSR | Rtemp1

-----  
 GOTO LX\_LR

End: ARM\_PC = ARM\_PC - 4

Le operazioni di load & store multiple del modo 4  
 10 contengono all'interno del loro opcode un campo di 16

bit che marca con un bit a livello alto i registri interessati al trasferimento.

Gli ultimi sedici bit dell'opcode vengono quindi esaminati uno ad uno e, per ogni bit a livello alto,  
 5 viene effettuata una operazione di load word o store word sul registro associato a quel bit.

Queste operazioni presentano quattro modi di indirizzamento:

- increment after (suffisso -IA);
- 10 - increment before (suffisso -IB);
- decrement after (suffisso -DA);
- decrement before (suffisso -DB).

Il registro base, se specificato dal bit 21 dell'opcode a livello alto, viene aggiornato alla fine  
 15 di ogni load o store singolo con il valore della successiva locazione puntata.

Occorre sottolineare che, sia che l'aggiornamento avvenga per decremento che per incremento, i registri con numero più alto sono associati agli indirizzi più  
 20 alti e quelli con numero più basso agli indirizzi più bassi.

Esistono poi versioni delle load & store multiple eseguibili solo in modo di funzionamento privilegiato che consentono di caricare dalla memoria il program  
 25 counter o di accedere ai registri di uso generale del modo User. Avendo mappato ogni registro del processore ARM, compresi quelli replicati, su un registro del processore LX, l'accesso ai registri del modo User è ottenibile in maniera immediata.

30 Il mapping dei registri del processore ARM sul processore LX, dettagliato in tabella 3, è gestito mappando i registri del modo corrente sui registri del processore LX da R16 a R31, in maniera tale che la traduzione delle operazioni risulti immediata. Per la  
 35 grande maggioranza delle istruzioni del processore ARM

infatti sono solo questi i registri accessibili ed il registro del processore LX è ricavabile dal registro del processore ARM specificato nell'opcode semplicemente sommando 16 al numero identificativo del registro.

I registri R13 ed R14 del processore ARM replicati per i modi Supervisor, Interrupt, Abort e Undefined sono mappati sui registri che vanno da R40 a R46, che serviranno come registri "di stack" per R29 e R30 quando il modo corrente è diverso da quello cui è associato il registro replicato.

Allo stesso modo R45 e R46 costituiscono i registri di stack per R13 e R14 per i modi User e System, così come R54 e R55 per il modo Fast Interrupt (FIQ).

I registri da R40 a R44 del processore LX costituiscono un'area di stack che conterrà i registri da R8 a R11 del modo User del processore ARM quando il modo corrente è Fast Interrupt, oppure i registri replicati del modo Fast Interrupt da R8 a R11 quando questo non è il modo corrente.

Le operazioni di load multiple possono anche caricare un dato dalla memoria nel program counter del processore ARM generando un salto e, nella versione 5 dell'instruction-set, aggiornare il registro CPSR con il registro SPSR del modo corrente. Le operazioni di load del program counter vengono quindi trattate in maniera diversa da quelle che interessano gli altri registri, anche perché occorre evitare che venga caricato in ARM\_PC un valore non word-aligned.

Va inoltre sottolineato che ogni operazione di store del program counter scrive in memoria il valore aggiornato, pari al PC dell'istruzione corrente aumentato di otto.

Tutte le operazioni del modo 3 effettuano solo accessi word-aligned, ignorando i due bit meno

significativi del registro base. Il valore aggiornato del registro base, tuttavia, verrà calcolato considerando anche questi ultimi due bit.

L'istruzione STM (multiple store) presenta due  
5 modalità di esecuzione:

- nel modo 1 l'istruzione è eseguibile sotto qualsiasi modo di lavoro del processore ARM e consente di salvare in locazioni consecutive di memoria un sottoinsieme qualunque dei registri del modo  
10 corrente;
- nel modo 2 l'istruzione è eseguibile solo in un modo privilegiato, mentre il suo effetto è imprevedibile nei modi User e System. Questa modalità consente di salvare in memoria un sottoinsieme qualunque dei  
15 registri dei modi User/System.

II processo di traduzione dell'istruzione STM, a parte la consueta valutazione della condizione di esecuzione, può essere suddiviso in tre fasi:

- una fase iniziale, in cui ottiene un indirizzo word-aligned mascherando i due bit meno significativi del registro base e, se la condizione di esecuzione non è verificata, si salta alla fine del programma. Nel modo 2 in questa fase si legge il registro di stato CPSR per capire se il modo di lavoro del processore  
20 ARM è User/System, FIQ (e quindi presenta più registri replicati) o un altro modo privileged;
- un ciclo che scandisce i 16 bit meno significativi dell'opcode e per ognuno di essi, in base al registro ed al modo di lavoro corrente, traduce la  
25 scrittura in memoria con l'eventuale swapping qualora l'endianness dei sistemi ARM ed LX differisca. L'ordine di scansione della lista dei registri, sia che l'aggiornamento degli indirizzi avvenga per decremento che per incremento, deve  
30 essere tale per cui i registri con numero più alto

vengano associati agli indirizzi più alti e quelli con numero più basso agli indirizzi più bassi;

- una fase finale, in cui si aggiorna il registro ARM\_PC, il cui contenuto non può essere modificato dall'istruzione STM, e se necessario si effettua il writeback del registro base. Sebbene durante l'accesso a memoria i due bit meno significativi dell'indirizzo debbano essere ignorati per generare accessi word-aligned, durante la fase di writeback bisogna tenerne conto.

Nella traduzione viene inserita una NOP (no operation) esplicita per dare tempo al core LX di valutare la condizione di esecuzione prima dell'eventuale salto. Il processore LX richiede infatti che tra la scrittura in un branch-bit e l'esecuzione del salto ad esso condizionato intercorra almeno un bundle di istruzioni.

La scansione dei bit della lista di registri interessati al trasferimento parte dal bit 0 ed arriva al bit 15 in caso di indirizzamento per incremento, mentre procede in senso opposto se l'indirizzamento è per decremento. A seconda che l'indirizzamento sia di tipo before o after, il valore del registro base deve essere aumentato (nel caso increment) o diminuito di 4 (nel caso decrement) prima o dopo aver effettuato la scrittura in memoria di ogni registro.

Nel modo 1 i registri da scrivere in memoria sono quelli associati al modo di lavoro corrente del processore ARM: questi registri sono sempre mappati sui registri di LX che vanno da R16 (ARM\_R0) a R31 (ARM\_R15 / ARM\_PC).

Se le endianness dei processori ARM ed LX sono uguali, non è necessario dedicare dei bundle allo swap.

La traduzione dell'istruzione precedente è valida anche per l'istruzione STM del modo 2 per i registri

non replicati in nessun modo privileged, ovvero ARM\_R15 e tutti i registri da ARM\_R0 ad ARM\_R7.

Al contrario, nel modo 2 per i registri da ARM\_R8 ad ARM\_R12 questa traduzione non è adatta. Questi  
 5 registri infatti sono replicati per il modo FIQ e quando il processore ARM entra in questa modalità i registri da R8 ad R12 del modo User/System sono salvati nei registri del processore LX che vanno da ARM\_R8stack ad ARM\_R12stack. La traduzione deve quindi iniziare  
 10 scegliendo quale registro scrivere in memoria tra ARM\_Rxstack (se la modalità corrente è FIQ) ed ARM\_Rx (in tutti gli altri casi).

L'ultimo caso da considerare è quello dell'istruzione STM di modo 2 per i registri R13 e R14.

15 In questo caso i registri sono replicati per ciascuno dei modi privilegiati e quindi, quando il modo corrente non è User o System, occorre salvare il registro ARM\_R13stack o ARM\_R14stack invece del corrispondente registro di modo corrente ARM\_R13 o  
 20 ARM\_R14.

La traduzione della istruzione STM termina con l'aggiornamento del registro ARM\_PC ed eventualmente con l'aggiornamento del registro base se l'istruzione richiede il writeback.

25 L'istruzione LDM (multiple load) presenta tre modalità di esecuzione differenti:

- nel modo 1 l'istruzione è eseguibile sotto qualsiasi modo di lavoro del processore ARM e consente di leggere da locazioni consecutive di memoria  
 30 scrivendo i dati letti in un sottoinsieme qualunque dei registri del modo corrente;
- nel modo 2 l'istruzione è eseguibile solo in un modo privilegiato, mentre il suo effetto è imprevedibile nei modi User e System. Questa modalità consente di  
 35 salvare i dati letti dalla memoria in un

sottoinsieme qualunque dei registri dei modi User/System. Il contenuto di ARM\_PC non può essere modificato;

- nel modo 3 l'istruzione è eseguibile solo in un modo privilegiato, mentre il suo effetto è imprevedibile nei modi User e System. Questa modalità consente di salvare i dati letti dalla memoria in un qualsiasi sottoinsieme dei registri di modo corrente che comprenda il registro ARM\_PC. Inoltre, il registro ARM\_3PSR del modo corrente è copiato in ARM\_CPSR. È necessario leggere i 5 LSB del registro di stato per individuare il modo di lavoro corrente e scegliere quale delle repliche del registro di stato SPSR salvare nel registro CPSR. Segue una complessa procedura che consente di effettuare lo switch del modo di lavoro corrente del processore ARM, che di fatto ricalca quasi perfettamente la traduzione dell'istruzione ARM MSR (move to status register from general-purpose register). Per la descrizione dettagliata di questa procedura si rimanda perciò alla successiva descrizione della traduzione dell'istruzione MSR.

Il processo di traduzione dell'istruzione LDM è analogo a quello della istruzione STM ma, per quanto detto in precedenza, deve rispettare più casi particolari.

Evidentemente, la scrittura sul program-counter del processore ARM può generare un salto nell'esecuzione del codice ARM ed il dispositivo traduttore deve essere preparato ad eseguirlo andando a puntare all'istruzione destinazione del salto.

La traduzione è del tutto analoga a quella della fase iniziale dell'istruzione STM, con l'unica differenza che il bundle di attesa del branch-bit che precede il salto condizionato è sfruttato per caricare

il link-register del processore LX in previsione del salto incondizionato da effettuare a fine traduzione.

Per i modi 1 e 3, che accedono ai soli registri del modo corrente, gli ultimi due bundle non sono  
5 necessari.

La scansione dei bit della lista di registri interessati al trasferimento segue esattamente la stessa regola usata per l'istruzione STM.

Anche in questo caso poi, a seconda che  
10 l'indirizzamento sia di tipo before o after, il valore del registro base deve essere aumentato (nel caso increment) o diminuito di 4 (nel caso decrement) prima o dopo aver effettuato la lettura in memoria ed il caricamento del valore letto in ogni registro.

Nei modi 1 e 3 i registri da scrivere sono quelli associati al modo di lavoro corrente del processore ARM: questi registri sono sempre mappati sui registri del processore LX che vanno da R16 (ARM\_R0) a R31 (ARM\_R15/ ARM\_PC).  
15

La traduzione precedente è valida anche per l'istruzione LDM del modo 2 per i registri non replicati in nessun modo privileged, ovvero quelli che vanno da ARM\_R0 ad ARM\_R7.  
20

Al contrario, nel modo 2 per i registri da ARM\_R8 ad ARM\_R12 questa traduzione non va bene. Questi registri infatti sono replicati per il modo FIQ e quando il processore ARM entra in questa modalità i registri da R8 ad R12 del modo User/System sono salvati nei registri LX che vanno da ARM\_R8stack ad  
25 ARM\_R12stack.  
30

La traduzione deve quindi, una volta realizzati l'accesso a memoria e l'eventuale swap, terminare scegliendo in quale registro scrivere il dato letto tra ARM\_Rxstack (se la modalità corrente è FIQ) ed ARM\_Rx  
35 (in tutti gli altri casi).

Si noti come, in questo caso, solo quattro dei bundle dedicati allo swap possono essere evitati se i sistemi hanno la stessa endianness. L'operazione di load word infatti ha latenza doppia rispetto alle  
 5 istruzioni LX di data-processing, e richiede perciò un bundle di attesa tra l'operazione di accesso a memoria e l'utilizzo del registro destinazione.

Una traduzione ancora diversa richiedono le scritture dei registri R13 ed R14 di modo User/System,  
 10 per l'istruzione LDM di modo 2. In questo caso i registri sono replicati per ciascuno dei modi privilegiati e quindi, quando il modo corrente non è User o System, occorre scrivere il registro ARM\_R13stack o ARM\_R14stack invece del corrispondente  
 15 registro di modo corrente ARM\_R13 o ARM\_R14.

Restano da analizzare i casi di scrittura del program-counter nei modi 1 e 3.

Nel primo caso è necessario effettuare il salto al valore scritto nel program-counter del processore ARM e  
 20 predisporre il dispositivo traduttore al salto forzandolo a puntare all'indirizzo destinazione del salto. La versione 5 dell'Instruction-Set del processore ARM richiede che la parola letta sia resa halfword-aligned e che il bit meno significativo della  
 25 parola stabilisca se entrare o meno in Thumb-state, impostando il bit 5 del registro di stato ARM\_CPSR.

Nel modo 3 invece, per la versione 5 dell'Instruction-Set del processore ARM, è necessario effettuare il salto al valore scritto nel program-  
 30 counter del processore ARM e predisporre il dispositivo traduttore al salto forzandolo a puntare all'indirizzo destinazione del salto.

Se il processore ARM sta lavorando in Thumb-state la parola letta deve essere resa halfword-aligned,  
 35 altrimenti deve essere resa word-aligned.

In entrambi i casi comunque il registro di stato ARM\_SPSR del modo corrente deve essere scritto nel registro di stato ARM\_CPSR.

La traduzione dell'istruzione LDM termina con  
 5 l'aggiornamento del registro ARM\_PC (che se è stato caricato da una operazione di load precedente è anche già stato incrementato di quattro) ed eventualmente con l'aggiornamento del registro base se l'istruzione richiede il writeback.

10 Se invece il registro ARM\_PC non è modificato e non è richiesto il writeback, la traduzione della fase finale diventa più semplice.

Il processore ARM presenta altre due istruzioni di accesso a memoria: SWP (swap word) e SWPB (swap byte).  
 15 Queste istruzioni effettuano ciascuna due accessi a memoria, caricando in un primo registro il contenuto di una locazione di memoria puntata da un registro base e scrivendo nella stessa locazione di memoria il contenuto di un secondo registro. Se il primo e il  
 20 secondo registro coincidono, si sono scambiati i contenuti del registro e della locazione di memoria.

L'istruzione SWP si comporta esattamente come una coppia di istruzioni LDR e STR, quindi deve tenere conto dell'endianness dei due sistemi ed effettuare  
 25 l'eventuale rotazione della word letta in base ai due bit meno significativi dell'indirizzo.

Si noti che se le endianness dei due sistemi sono diverse le operazioni di swap da fare sono due: una sulla word letta dall'operazione di load ed una prima  
 30 di effettuare l'operazione di store.

Si sottolinea che per il processore ARM tutte le operazioni di swap che coinvolgono il program-counter, sia come operando che come registro base, sono imprevedibili.

35 L'istruzione SWPB non presenta problemi di

endianness ed ha una traduzione molto più semplice, con l'avvertenza di separare le due ultime istruzioni della traduzione, per fare in modo che nel primo bundle della traduzione della successiva istruzione ARM si  
 5 possa accedere al registro di destinazione Rdest senza dare luogo a read-hazard.

Le istruzioni LX per la lettura di un byte in memoria (LDB e LDBU) richiedono infatti che si attenda per due bundle prima di accedere al byte letto.

10 Il processore ARM presenta poi tre istruzioni di salto:

- salto condizionato PC-relative (con e senza memoria dell'indirizzo di ritorno): l'offset di 24 bit è contenuto nell'opcode del salto. Per calcolare  
 15 l'indirizzo di destinazione esso viene moltiplicato per quattro (in quanto ogni opcode ARM occupa 32 bit) ed esteso con segno, per poi essere sommato al valore attuale del program counter. È opportuno sottolineare che, in conseguenza dell'architettura  
 20 della pipeline del processore ARM, al momento dell'aggiornamento che avviene nella fase di esecuzione, il program counter contiene l'indirizzo dell'istruzione di salto incrementato di otto;
- salto incondizionato con cambiamento di modo: il  
 25 processore effettua un salto con offset di 24 bit, mantiene memoria dell'indirizzo di ritorno nel link register ed entra in modo Thumb, modificando il T bit della parola di stato;
- salto condizionato con cambiamento di modo (con o  
 30 senza memoria dell'indirizzo di ritorno): il processore effettua un salto all'indirizzo contenuto in un registro indice. Il valore del registro indice viene allineato trascurandone il bit meno significativo, che viene utilizzato per decidere il  
 35 modo di funzionamento (se a livello alto modo Thumb,

altrimenti modo ARM).

- 5 Sia i salti ad offset PC-relative che quelli da registro vengono tradotti su LX con una operazione di salto a link, anche in conseguenza del fatto che il processore ARM può realizzare salti PC-relative più lunghi del processore LX, i cui opcodes contengono un campo che fornisce l'offset di un bit più corto rispetto ad ARM.

- 10 L'istruzione ARM B(branch) effettua un salto condizionato PC-relative senza tenere memoria dell'indirizzo di ritorno nel link-register.

L'offset contenuto nell'opcode deve essere esteso con segno e moltiplicato per due per ottenere l'offset espresso come numero di byte.

- 15 La sua traduzione comincia con la consueta valutazione della condizione di esecuzione e continua nel modo seguente:

Istruzione ARM	Traduzione LX
B@@ signed_offset	Condition Evaluation
	Rt_dest = ARM_PC +
	#byte_offset
	ARM_PC = ARM_PC-4
	-----
	ARM_PC = (\$Condition)?
	Rt_dest : ARM_PC
	LX_LR = (\$Condition)?
	Rt_dest : ARM_PC
Byte_offset = signed_offset	-----
« 2	Rshift_op = ARMPOINTER_ADDR
	-----
	MemoryWord(Rshift_op) =
	LX_LR
	-----
	GOTO LX_LR

L'istruzione BL (branch and link) effettua un salto

condizionato PC-relative, tenendo memoria dell'indirizzo di ritorno nel link-register del processore ARM (R14).

5 L'istruzione BX (branch and exchange to Thumb) effettua un salto condizionato all'indirizzo contenuto in un registro target, senza tenere memoria dell'indirizzo di ritorno.

10 Il valore contenuto nel registro target Rtarget va reso halfword-aligned ed il suo bit meno significativo, scartato durante l'operazione di allineamento, va a modificare il T-bit del registro di stato ARM\_CPSR. Portando il T-bit a livello alto il processore entra nella modalità Thumb.

15 L'istruzione BLX (branch, link and exchange to Thumb) del modo 2 è identica alla precedente BX ma tiene memoria dell'indirizzo di ritorno nel link-register del processore ARM.

20 Esiste anche un'altra versione dell'istruzione BLX che effettua un salto PC-relative e che viene denominata modo 1.

25 Questa istruzione non supporta l'esecuzione condizionale e contiene all'interno dell'opcode un offset immediato a 24 bit che va moltiplicato per quattro, esteso con segno e quindi sommato al valore corrente del program counter.

Il bit 24 (H bit) dell'opcode va moltiplicato per due e sommato al valore aggiornato del program-counter per ottenere un indirizzo destinazione comunque halfword-aligned.

30 Il processore ARM deve sempre entrare in Thumb-state.

35 Il processore ARM presenta inoltre due istruzioni dedicate alla manipolazione dei registri di stato che consentono la lettura e la scrittura dei registri di stato CPSR e SPSR associati al modo corrente.

Se l'istruzione ha come sorgente o destinazione SPSR, dal momento che tutti i modi di funzionamento del processore ARM, tranne il modo User ed il modo System, hanno uno SPSR replicato, la prima cosa da fare è  
 5 individuare il modo corrente sulla base del contenuto del registro LX che emula il registro CPSR del processore ARM.

Non volendo accedere direttamente alle risorse del core LX, si individua il modo corrente (descritto dai  
 10 cinque bit meno significativi del CPSR) tramite una serie di operazioni di confronto che impostano un diverso branch-bit di LX per ogni modo di lavoro del processore ARM. Per le operazioni di lettura dello stato (MRS), a questo punto mediante una serie di  
 15 operazioni di select (SLCT) si decide cosa scrivere nel registro destinazione. Per le operazioni di scrittura (MSR), sempre attraverso una serie di select, si aggiorna solo il valore del SPSR associato allo stato corrente mentre gli altri vengono lasciati invariati.

20 L'accesso al registro CPSR ovviamente non presenta questo problema, ma la sua scrittura può forzare un cambiamento del modo di lavoro del processore ARM. Il mapping dei registri del processore ARM su LX è stato descritto in precedenza ed è rappresentato nella  
 25 tabella 3.

Le operazioni MSR (move to status-register from register) modificano con un' immediato o con il contenuto di un registro sorgente uno o più dei byte componenti un registro di stato.

30 I byte da modificare sono individuati dalla maschera che occupa i bit dal 16 al 19 dell'opcode: per ogni bit alto della maschera il corrispondente byte della parola di stato viene modificato.

Si considera inizialmente il caso di  
 35 indirizzamento da registro, con CPSR come destinazione.

La traduzione di questa istruzione prevede i seguenti passi, che devono essere eseguiti da tutte le istruzioni che possono cambiare il modo di lavoro del processore ARM :

- 5        1) si determina il modo di lavoro corrente del processore ARM mediante una serie di operazioni di confronto che impostano un diverso branch-bit per ogni modo e si mascherano il registro sorgente ed il registro CPSR con due maschere complementari.
- 10       La scrittura sul registro CPSR in modo User deve essere ignorata.

Istruzione ARM

MSR CPSR\_<fields>, Rsorg

Traduzione LX

Condition Evaluation

-----

NOP

-----

Rtemp1 = ARM\_CPSR & 0x01F

Rtemp2 = Rsorg & #field\_mask

IF (! \$Condition) GOTO end

-----

Field\_mask        =        Mask  
<fields>)

( \$IsUSR= (Rtemp1==16)

\$IsSYS= (Rtemp1==31)

\$IsFIQ = (Rtemp1 ==17)

\$IsSPV= (Rtemp1 ==19)

-----

\$IsIRQ=(Rtemp1==18)

\$IsUND = (Rtemp1 == 27)

\$IsABT = (Rtemp1 == 23)

-----

Rtemp1    =    ARM\_CPSR    &    (~  
#field\_mask)

IF (\$IsUSR) GOTO end

-----

- 2) si scambiano i contenuti dei registri da R8 a R12 del processore ARM con quelli corrispondenti della

zona di stack, per prepararsi ad una eventuale transizione al modo FIQ

Istruzione ARM

Traduzione LX

MSR CPSR\_<fields>, Rsorg

...

-----

ARM\_R8 = ARM\_R8stack

ARM\_R8stack = ARM\_R8

ARM\_R9 = ARM\_R9stack

ARM\_R9stack = ARM\_R9

-----

ARM\_R10 = ARM\_R10stack

ARM\_R10stack = ARM\_R10

ARM\_R11 = ARM\_R11stack

ARM\_R11stack = ARM\_R11

-----

ARM\_R12 = ARM\_R12stack

ARM\_R12stack = ARM\_R12

IF (\$IsSPV) GOTO spv\_proc

-----

- 3) individuato il modo corrente, si salvano i valori dei registri R13 e R14 nei corrispondenti registri replicati

Istruzione ARM

Traduzione LX

MSR CPSR\_<fields>, Rsorg

-----

IF (\$IsIRQ) GOTO irq\_proc

-----

IF (\$IsUND) GOTO und\_proc

-----

IF (\$IsABT) GOTO abt\_proc

-----

IF (\$IsFIQ) GOTO fiq\_proc

Spv\_proc: ARM\_R13spv =

ARM\_R13

ARM\_R14spv = ARM\_R14

GOTO continue

```

-----
irq_proc:      ARM_R13irq      =
ARM_R13
ARM_R14irq = ARM_R14
GOTO continue
-----
und_proc: ARM_R13und=ARM_R13
ARM_R 14und = ARM_R14
GOTO continue
-----
abt_proc:      ARM_R13abt      =
ARM_R.13
ARM_R14abt == ARM_R14
GOTO continue
-----
fiq_proc:      ARM_R13fiq      =
ARM_R13
ARM_R14fiq = ARM_R14
-----

```

4) si aggiornano il registro CPSR ed i registri dei  
flag

Istruzione ARM

MSR CPSR\_<fields>, Rsorg

Traduzione LX

...

```

-----
continue: ARM_CPSR = Rtemp1
| Rtemp2
-----

```

```

Rtemp1 = ARM_CPSR & 0x01F
-----

```

```

RtV = ARM.CPSR » 28

```

```

RtC = ARM_CPSR » 29

```

```

RtZ = ARM_CPSR » 30
-----

```

```

RN = ARM_CPSR » 31

```

```

RV = RtV & 0x01

```

RC = RtC & 0x01

RZ=RtZ&0x01

...

5) si determina il nuovo modo di lavoro del processore ARM come al punto 1

Istruzione ARM

Traduzione LX

MSR CPSR\_<fields>, Rsorg

\$IsFIQ = (Rtemp1 == 17 )

-----

\$IsSPV = (Rtemp1 == 19 )

\$IsIRQ = (Rtemp1 == 18 )

Rtemp6 = (Rtemp1 == 31 )

Rtemp5 = (Rtemp1 == 16 )

-----

\$IsUND = (Rtemp1 == 27 )

\$IsABT = (Rtemp1 == 23 )

\$IsUNPRV = (Rtemp6 == 1) ||

(Rtemp5 == 1)

IF (\$IsFIQ) GOTO get\_fiq

-----

6) se non c'è stato un passaggio al modo FIQ  
riscambio i valori dei registri che sono stati spostati  
al punto 2)

Istruzione ARM

Traduzione LX

MSR CPSR\_<fields>, Rsorg

...

-----

ARM\_R8 = ARM\_R8stack

ARM\_R8stack = ARM\_R8

ARM\_R9 = ARM\_R9stack

ARM\_R9stack = ARM\_R9

-----

ARM\_R10 = ARM\_R10stack

ARM\_R10stack = ARM\_R10

ARM\_R11 = ARM\_R1 Istack

ARM\_R11stack = ARM\_R11

7) si scrivono in R13 e R14 il contenuto dei registri replicati associati al nuovo modo

Istruzione ARM

Traduzione LX

MSR CPSR\_<fields>, Rsorg

...

-----  
ARM\_R12 = ARM\_R12stack  
ARM\_R12stack = ARM\_R12  
...  
IF\$isSPV) GOTO get.spv  
-----

IF (\$IsIRQ) GOTO getJrq  
-----

IF (\$IsUND) GOTO get\_und  
-----

IF (\$IsABT) GOTO get\_abt  
-----

ARM\_R13 = ARM\_R13stack

ARM\_R14 = ARM\_R14stack  
-----

get\_spv: ARM\_R13 =

ARM\_R13spv

ARM\_R14 = ARM\_R14spv

GOTO end  
-----

get\_rq: ARM\_R13 = ARM\_R13irq

ARM\_R14 = ARM\_R14irq

GOTO end  
-----

get\_und: ARM\_R13 =

ARM\_R13und

ARM\_R14 = ARM\_R14und

GOTO end  
-----

```

get_abt:      ARM_R.13      =
ARM_R13abt
ARM_R14 = ARM_R14abt
GOTO end
-----
get_fiq:      ARM_R8        =
ARM_R8stack
ARM_R8stack = ARM_R8
ARM_R9 = ARM_R9stack
ARM_R9stack = ARM_R9
-----
ARM_R10 = ARM_R10stack
ARM_R10stack = ARM_R10
ARM_R11 = ARM_R11stack
ARM_R11stack = ARM_R11
-----
ARM_R12 = ARM_R12stack
ARM_R12stack = ARM_R12
ARM_R13 = ARM_R13fiq
ARM_R14 = ARM_R14fiq
-----
end: ARM_PC = ARM_PC - 4

```

L'istruzione MSR che scrive nel registro CPSR un immediato ha una traduzione del tutto analoga alla precedente.

- Quando invece il registro destinazione  
 5 dell'istruzione MSR è il registro SPSR del modo  
 corrente, la traduzione cambia perchè occorre  
 individuare lo stato corrente e nel frattempo preparare  
 gli aggiornamenti degli SPSR per i vari modi (l'accesso  
 al SPSR nei modi User e System rende imprevedibile  
 10 l'esecuzione).

Alla fine viene aggiornato solo il registro SPSR del modo corrente, tramite l'istruzione MSR\_SPSR.

L'operazione MRS (move to register from status-

register) che legge il registro CPSR necessita di ricostruire il contenuto informativo del CPSR stesso, che in questa implementazione è distribuito tra ARM\_CPSR ed i quattro registri dei flag RC, RN, RZ, RV.

5 L'istruzione supporta l'esecuzione condizionale.

L'operazione MRS (move to register from status-register) che legge il registro SPSR del modo corrente deve prima leggere il registro CPSR per individuare il modo corrente e la sua esecuzione ha risultato  
10 imprevedibile nei modi User e System che non hanno un registro SPSR.

Oltre alle istruzioni già descritte, il processore ARM presenta altre istruzioni speciali:

- BKPT (software breakpoint );
- 15 - SWI (software interrupt);
- istruzioni per la gestione dei coprocessori (load from coprocessor, store to coprocessor, coprocessor data-processing, ecc.)

L'istruzione di breakpoint software serve solamente  
20 in fase di debugging e pertanto non deve essere presente in nessun file eseguibile per un processore ARM. Di conseguenza è stata tradotta sul processore LX, come anche tutti gli opcode che risultano non definiti, in una system call del tipo Illegal Instruction.

25 Allo stesso modo sono state trattate le operazioni sui coprocessori, non essendo nei nostri attuali obiettivi l'emulazione completa di un sistema hardware basato sul processore ARM.

Le interruzioni software consentono al processore  
30 ARM di interagire con l'hardware del sistema e possono essere tradotte in tre modi:

- come Illegal Instruction, se non si vuole emulare l'hardware del sistema;
- come salto ad un apposito ARM exception-handler  
35 scritto per il processore LX, che si occupa di

chiamare la system-call del processore LX corrispondente al servizio di software interrupt invocato dal processore ARM. Questa soluzione può essere attuata in fase di sviluppo del sistema misto ARM-LX per verificare la corretta esecuzione dei programmi;

- avendo a disposizione il codice di tutti gli exception handlers ed il contenuto dei vettori di interrupt, l'istruzione SWI può essere tradotta semplicemente come passaggio al modo Supervisor del processore ARM e salto con link al vettore degli interrupt. Le operazioni di gestione dello stato e di salvataggio e ripristino dei registri sono infatti presenti esplicitamente nel codice ARM e non sono realizzate dall'opcode SWI. Questa scelta consente l'emulazione di un sistema completo basato sul processore ARM, ma per funzionare correttamente necessita di un sistema di memoria che realizzi una partizione tra memoria ARM e memoria LX, in cui nella memoria ARM saranno mappate le periferiche del processore ARM che non hanno un equivalente nel sistema LX e in cui gli accessi a periferiche già presenti nel sistema LX siano reindirizzati alle corrispondenti locazioni della memoria del processore LX.

Da quanto sopra descritto, risulta evidente che la traduzione degli opcode del processore ARM sul processore LX ha un impatto pesante sulle prestazioni del sistema, ad esempio nella traduzione di una istruzione di data-processing che non modifica il registro di stato.

Per ogni istruzione ARM appartenente a questa categoria occorrono almeno quattro bundle del processore LX, quindi a pari frequenza di clock l'esecuzione del codice ARM sul processore LX rallenta

di quattro volte.

Si ha un peggioramento ulteriore per le istruzioni di moltiplicazione, di accesso a memoria e per i salti.

Da una analisi sull'esecuzione di alcuni benchmark  
5 scritti in codice ARM emergono alcune osservazioni di rilievo:

- le operazioni di data-processing sono mediamente il 50% del totale delle istruzioni eseguite;
- oltre il 90% delle istruzioni non sfruttano l'esecuzione condizionale;
- delle operazioni di data-processing, circa il 90% si suddivide in due modi di indirizzamento: quello diretto da registro e quello con immediato non ruotato;
- 15 - delle operazioni di data-processing, meno del 20% richiede di modificare il registro di stato CPSR;
- le istruzioni che modificano il registro CPSR sono nella gran parte dei casi operazioni di confronto (CMN, CMP) o di test logico (TST, TEQ).

20 In conseguenza di tutto ciò, risulta conveniente complicare leggermente la fase di decodifica per aggiungere traduzioni ad hoc per le istruzioni più utilizzate.

Questa modalità di traduzione veloce e' applicabile quando tra gli operandi dell'istruzione non e' presente il program-counter del processore ARM.

Un'istruzione non condizionale di data-processing, se non modifica il registro di stato, viene tradotta in questo modo:

30 - se l'indirizzamento è diretto da registro

Istruzione ARM	Traduzione LX
ORR@@ Rdest, Rsorg1, @@@	Rdest=Rsorg1 Rsorg2
	ARM_PC=ARM_PC+4

se l' indirizzamento è da immediato non ruotato

Istruzione ARM	Traduzione LX
----------------	---------------

```

ORR Rdest, Rsorg1, #short_imm Rdest = Rsorg1 | #short_imm
                                ARM_PC = ARM_PC + 4

```

Un'istruzione non condizionale di test logico o di confronto viene tradotta in questo modo:

- se l'indirizzamento è diretto da registro

Istruzione ARM	Traduzione LX
CMP Rsorg1.Rsorg2	Rdest = Rsorg1 - Rsorg2
	\$Condition == 1
	ARM_PC = ARM_PC + 8
	-----
	Commitment

se l' indirizzamento è da immediato non ruotato

Istruzione ARM	Traduzione LX
CMP Rsorg, #short_imm	Rt_dest = Rsorg - #short_imm
	Rshift_op = #short_imm
	\$Condition = 1
	ARM_PC = ARM_PC + 8
	-----
	Commitment

- 5 La fase di commitment avviene esattamente come descritto nel paragrafo precedente, ed e' per questo che il branch-bit Condition viene portato ad uno ed ARM\_PC aumentato di otto.

- 10 Con questa modifica la gran parte delle istruzioni di data-processing può' essere eseguita in un singolo bundle.

La soluzione appena descritta consente di conseguire notevoli vantaggi rispetto alle soluzioni note.

- 15 Si apprezzerà che il principale vantaggio della soluzione secondo l'invenzione deriva dal fatto che l'introduzione di un dispositivo traduttore esterno permette di lasciare inalterato il core del microprocessore LX. Detto dispositivo traduttore,
- 20 quando necessita di accedere alle risorse del core del

microprocessore LX, non lo accede direttamente, ma incorpora nella traduzione dell'istruzione ARM dei costrutti condizionali basati sul contenuto dei registri o dei branch bit del core del microprocessore LX.

Inoltre vantaggiosamente il dispositivo traduttore entra in azione autonomamente, riconoscendo gli accessi alla zona di memoria riservata al codice ARM.

Gli esperti del settore apprezzeranno che la soluzione qui descritta con specifico riferimento alla traduzione di istruzioni ARM in istruzioni ST-200 LX è in realtà applicabile ad un campo di impiego più ampio, ossia alla traduzione delle istruzioni di un microprocessore scalare pipelined avente caratteristiche comunque riconducibili alle caratteristiche di un processore ARM verso un microprocessore del tipo VLIW. avente caratteristiche comunque riconducibili alle caratteristiche di un processore LX. Questo concetto è stato espresso nelle rivendicazioni che seguono facendo riferimento rispettivamente a processori "di tipo ARM" e di "tipo LX". Si osserva a tal proposito che la soluzione descritta è applicabile anche ad un processore superscalare, il quale supporta renaming e esecuzione fuori ordine, rendendo possibili ottime prestazioni anche su traduzioni non perfettamente ottimizzate.

Si apprezzerà ancora che tali rivendicazioni fanno di per sé riferimento a un primo insieme di istruzioni - eseguibili - su un processore di tipo ARM e ad un secondo insieme di istruzioni - eseguibili - su un processore di tipo LX. La menzione di tali tipi di processori è quindi destinata in via primaria ad identificare le caratteristiche di tali insiemi di istruzioni, non diversamente definibili.

Le istruzioni di tipo ARM e di tipo LX identificano in generale tutti i processi che coinvolgono architetture di set di istruzioni (ISA) equivalenti a quella descritta.

- 5 Naturalmente, fermo restando il principio dell'invenzione, i particolari di realizzazione e le forme di attuazione potranno essere ampiamente variati rispetto a quanto descritto ed illustrato, senza per questo uscire dall'ambito della presente invenzione,
- 10 così come definita dalle rivendicazioni annesse.

\* \* \*

### RIVENDICAZIONI

1. Procedimento per tradurre istruzioni appartenenti a un primo insieme di istruzioni eseguibili su un processore di tipo ARM in istruzioni  
5 appartenenti a un secondo insieme di istruzioni eseguibili su un processore di tipo LX comprendente un nucleo o core, caratterizzato dal fatto di comprendere le seguenti operazioni:
- prevedere un primo insieme di registri  
10 corrispondenti alle istruzioni di detto primo insieme,  
prevedere un secondo insieme di registri corrispondente alle istruzioni di detto secondo insieme,  
mappare ciascun registro di detto primo insieme in  
15 un corrispondente registro di detto secondo insieme destinato a emularne il comportamento realizzando una traduzione univoca indipendente dei dati di detto primo insieme di istruzioni in detto secondo insieme di istruzioni,
- 20 dette operazioni di provvedere un secondo insieme di registri e di mappare essendo ottenute aggiungendo unità funzionali (10) a detto nucleo (14), che è mantenuto inalterato  
realizzare detta traduzione in assenza di accesso  
25 diretto alle risorse di detto nucleo o core (14).
2. Procedimento secondo la rivendicazione 1, caratterizzato dal fatto che dette operazioni di provvedere un secondo insieme di registri e di mappare sono ottenute aggiungendo a detto core unità funzionali  
30 comprendenti un dispositivo di traduzione (10) esterno a detto nucleo (14) del processore di tipo LX,  
detto dispositivo di traduzione (10) eseguendo l'operazione di intercettare gli accessi a un'area di memoria (17) riservata alle istruzioni del primo  
35 insieme.

3. Procedimento secondo la rivendicazione 2, caratterizzato dal fatto che comprende inoltre le operazioni di forzare un contatore di programma del processore di tipo LX a puntare a una memoria di traduzione (11) riservata nel dispositivo di traduzione (10) per contenere la traduzione (T) dell'istruzione appartenente al primo insieme di istruzioni che nel frattempo viene decodificata.

4. Procedimento secondo una o più delle rivendicazioni precedenti, caratterizzato dal fatto di prevedere di caricare in detta memoria di traduzione (11) riservata tutte le istruzioni che costituiscono la traduzione dell'istruzione del primo insieme decodificata.

5. Procedimento secondo una o più delle rivendicazioni precedenti, caratterizzato dal fatto di associare inoltre a dette istruzioni che costituiscono la traduzione (T) dell'istruzione del primo insieme decodificata un salto a link alla prossima istruzione del primo insieme da eseguire, in modo che tutte le istruzioni del primo insieme non direttamente mappabili su una istruzione del secondo insieme necessitino di un salto all'area di memoria riservata (11) e di un salto a link per caricare la successiva istruzione del primo insieme.

6. Procedimento secondo una o più delle rivendicazioni precedenti, caratterizzato dal fatto di tradurre tutte le istruzioni che non trovano una singola istruzione del secondo insieme di istruzioni equivalente in un salto incondizionato di tipo GOTO.

7. Procedimento secondo una o più delle rivendicazioni precedenti, caratterizzato dal fatto di comprendere l'operazione di forzare il contatore di programma associato al processore di tipo LX a emulare il funzionamento di un contatore di programma associato

al processore di tipo ARM.

8. Procedimento secondo la rivendicazione 7, caratterizzato dal fatto che detta operazione di forzare il contatore di programma associato al processore di tipo LX prevede che detto contatore di programma debba contenere lo stesso valore del contatore di programma associato al processore di tipo ARM all'atto del caricamento di un'istruzione appartenente al primo insieme di istruzioni.

9. Procedimento secondo la rivendicazione 8, caratterizzato dal fatto che al termine dell'esecuzione emulata viene eseguito un salto all'indirizzo della prossima istruzione appartenente al primo insieme di istruzioni.

10. Procedimento secondo la rivendicazione 9, caratterizzato dal fatto che prevede di emulare il contatore di programma del processore di tipo ARM tramite un registro contatore (ARM\_R15/ARM\_PC) e di incrementare detto registro contatore (ARM\_R15/ARM\_PC) in modo che ogni istruzione che acceda a detto registro contatore (ARM\_R15/ARM\_PC) durante la fase di esecuzione abbia un comportamento coerente con l'esecuzione sul processore di tipo ARM e di decrementare detto registro contatore (ARM\_R15/ARM\_PC) per puntare alla successiva istruzione appartenente al primo insieme di istruzioni.

11. Procedimento secondo la rivendicazione 10, caratterizzato dal fatto che detto registro contatore (ARM\_R15/ARM\_PC) è incrementato di un valore otto e decrementato successivamente di un valore quattro.

12. Procedimento secondo la rivendicazione 11, caratterizzato dal fatto che per le istruzioni appartenenti al primo insieme di istruzioni che hanno come destinazione il contatore di programma il puntamento alla successiva istruzione avviene caricando

in un link register del processore di tipo LX il valore aggiornato del registro contatore (ARM\_PC) ed effettuando un salto incondizionato del tipo GOTO link.

5     13. Procedimento secondo una delle rivendicazioni da 2 a 6, caratterizzato dal fatto che prevede di lasciare evolvere liberamente il contatore di programma associato al processore di tipo LX in assenza di salti.

10     14. Procedimento secondo la rivendicazione 13, caratterizzato dal fatto che detto dispositivo di traduzione (10) è atto a eseguire le operazioni di intercettare gli accessi alla zona di memoria riservata (17) alle istruzioni del primo insieme e a controllare un insieme di registri puntatori () per decidere se eseguire le prossime istruzioni come istruzioni del  
15     secondo insieme o come istruzioni del primo insieme da emulare.

20     15. Procedimento secondo la rivendicazione 14, caratterizzato dal fatto che detto dispositivo di traduzione (10) è inattivo finchè il nucleo del processore di tipo LX esegue istruzioni appartenenti a detto secondo insieme e rimanda gli accessi a memoria a dei dispositivi sottostanti, in particolare a una memoria cache di istruzioni () e che detto dispositivo di traduzione si attiva (10) quando si verifica un  
25     accesso alla zona di memoria (17) riservata a contenere il primo insieme di istruzioni.

30     16. Procedimento secondo la rivendicazione 15, caratterizzato dal fatto che quando detto dispositivo di traduzione si attiva, carica in un suo registro interno (NEXT\_ARM\_INSTR) appartenente all'insieme di registri puntatori () l'indirizzo a cui si accede ed effettua la lettura dell'istruzione dalla zona di memoria (17) corrispondente.

35     17. Procedimento secondo la rivendicazione 16, caratterizzato dal fatto che prevede inoltre le

operazioni di tradurre detta istruzione letta dall'area di memoria riservata (17) e di memorizzare detta istruzione

5 di allocare una finestra di esecuzione alla quale vengono rimandati tutti gli accessi ad indirizzi di memoria che partono dal valore corrente del contatore di programma del processore di tipo LX e coprono un'area pari a quella occupata dalla traduzione,

10 di leggere tramite il nucleo del processore di tipo LX la prima istruzione della traduzione dall'area di memoria riservata (11) nel dispositivo di traduzione

di incrementare il registro (NEXT\_ARM\_INSTR), in particolare di un valore quattro, per puntare alla prossima istruzione del primo insieme di istruzioni,

15 di chiudere detta finestra di esecuzione dopo la lettura dell'ultima istruzione del primo insieme da tradurre e, se al successivo accesso in memoria il registro (NEXT\_ARM\_INSTR) punta al di fuori della zona di memoria riservata (11), disattivare il dispositivo di traduzione (10).

20 18. Procedimento secondo la rivendicazione 17, caratterizzato dal fatto che prevede inoltre le operazioni di prevedere in presenza di salti nel programma costituito dalle istruzioni del primo insieme di riscrivere detto registro (NEXT\_ARM\_INSTR) mediante una operazione di store word contenuta nella traduzione in istruzioni del secondo insieme dell'istruzione del primo insieme.

30 19. Procedimento secondo una delle rivendicazioni precedenti caratterizzato dal fatto che l'operazione di traduzione sul processore di tipo LX inizia con la verifica della condizione di esecuzione, che consiste nella valutazione di uno o più dei flag presenti nel registro di stato (CPSR).

35 20. Dispositivo di traduzione di istruzioni

appartenenti a un primo insieme di istruzioni eseguibili su un processore di tipo ARM in istruzioni appartenenti a un secondo insieme di istruzioni eseguibili su un processore di tipo LX comprendente un

5 nucleo o core, caratterizzato dal fatto che detto dispositivo di traduzione (10) comprende un sottosistema di traduzione (12) atto a ricevere in ingresso un'istruzione (IA) del primo insieme e fornire in uscita una traduzione (T) comprendente una o più

10 istruzioni del secondo insieme, una memoria di traduzione (11) per memorizzare detta traduzione (T), un dispositivo di controllo (13) per prelevare detta traduzione (T) da detta memoria di traduzione (11) e fornirla al nucleo (14) di detto processore di tipo LX.

15 21. Dispositivo secondo la rivendicazione 20 caratterizzato dal fatto che detto sottosistema di traduzione (12) opera sulla base di una tabella di codici memorizzata in detta memoria di traduzione (11)

20 22. Dispositivo secondo la rivendicazione 21 caratterizzato dal fatto che è disposto connesso fra il nucleo (14) di detto processore di tipo LX e una memoria cache di istruzione (16) di detto processore di tipo LX operante su prima memoria (17) contenente istruzioni del primo insieme e una seconda memoria

25 (18) contenente istruzione del secondo insieme e che il dispositivo di controllo (13) intercetta gli accessi (A1,D1) del nucleo del processore di tipo LX (14) a dette memorie (17, 18).

30 23. Dispositivo secondo la rivendicazione 22 caratterizzato dal fatto che comprende un insieme di registri puntatori (15) almeno in parte atti a controllare l'accesso a dette memorie (17,18).

35 24. Prodotto informatico direttamente caricabile nella memoria di un elaboratore numerico e comprendete porzioni di codice software per attuare il procedimento

secondo una qualsiasi delle rivendicazioni da 1 a 19  
quando il prodotto è eseguito su un elaboratore.

### RIASSUNTO

Un procedimento per tradurre istruzioni appartenenti a un primo insieme di istruzioni eseguibili su un processore di tipo ARM in istruzioni  
5 appartenenti a un secondo insieme di istruzioni eseguibili su un processore di tipo LX comprendente un nucleo o core, provvede un primo insieme di registri corrispondenti alle istruzioni eseguibili su processore di tipo ARM e un secondo insieme di registri  
10 corrispondente alle istruzioni eseguibili su processore di tipo LX.

Ciascun registro di detto primo insieme viene mappato in un corrispondente registro di detto secondo insieme destinato a emularne il comportamento  
15 realizzando una traduzione univoca indipendente dei dati di detto primo insieme di istruzioni in detto secondo insieme di istruzioni. Detta traduzione è attuata tramite un dispositivo di traduzione esterno al core del processore di tipo LX, detto core rimanendo  
20 dunque inalterato, in particolare con riferimento alla sua issue logic, e la traduzione operando in assenza di accesso alle risorse di detto core, tramite intercettazione degli accessi del core alla zona di memoria riservata alle istruzioni ARM da parte del  
25 dispositivo di traduzione.

Applicazione preferenziale in un microprocessore ST-200 LX.

(Figura 1).

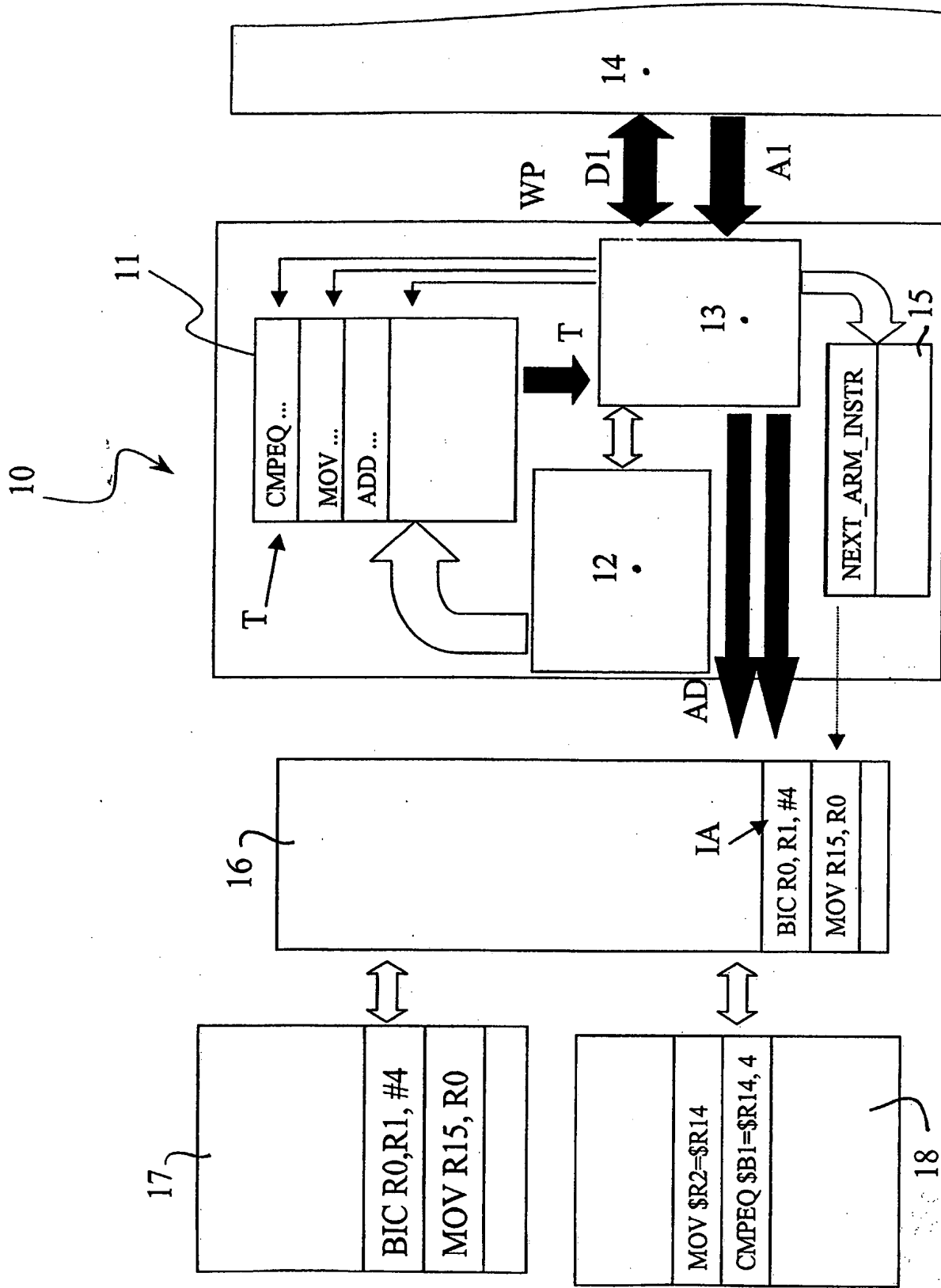


Fig. 1